Management of the Internet and Complex Services

European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D6.1

Open source support & joint software development interim report

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
International University Bremen, IUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universitat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, CETIM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University of Surrey, UniS, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois — Campus scientifique
615, rue de Jardin Botanique — B.P. 101
F—54600 Villers Les Nancy Cedex
France
Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title: EMANICS Specification of the static & dynamic content for dissemination environment + Approved and operational Web site

Type: Public

Editor(s): Olivier Festor

E-mail: Olivier.Festor@loria.fr

Author(s): Mark Burgess, Vincent Cridlig, Olivier Festor, Robert Szuman, Emil Lupu, George Pavlou, Juergen Schoenwaelder

Doc ID: D6_1-v1_0.doc

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.1	September 26, 2006	Olivier Festor	First version, providing the ToC
V0.2	October 12, 2006	Robert Szuman	Open Source Inventory section
V0.3	October 14, 2006	Olivier Festor	ENSUITE part + Introduction
V0.4	October 20,2006	George Pavlou	OSIMIS Part
V0.5	october 21, 2006	Emil Lupu	Ponder section
V1.0	October 30, 2006	Olivier Festor	Conclusion & typos

Legal Notices

Table of Contents

(This page is left blank intentionally.)

Executive Summary

Open Source support and coherent joint software development is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need to support the Open Source initiatives in the area of management to foster again their use and acceptance on large scale.

This document is an interim report of the activity undertaken in work-package 6 for increasing the availability, visibility and acceptance of Open Source software in the area of device, network and service management. It describes (1) the efforts made in collecting data on available software, making this data available to the public in an homogeneous and open way and (2) the efforts supported by EMANICS on the development and/or extensions towards broader acceptance and dissemination of Open Source components that emerge from the EMANICS community.

1 Introduction

Open Source support and coherent joint software development is a very important initiative for ensuring strong integration, network survivability and transfer of technology developed within the network. In the management community, many Open Source components have been produced over the years, some of them being widely used today for several purposes. Recent evolutions in management and networking technologies however did slow the use of these very valuable software components and the rhythm of new productions has decreased drastically. From the very beginning, EMANICS partners did recognize the need support the Open Source initiatives in the area of management to foster again their use and acceptance on large scale.

The goal of the WP6 work package is to run the tasks related to these Open Source related activities. Its objectives are:

- Promote the use of open source software for device, network and service management purpose,
- Contribute to the identification and evolution of open-source software emerging within the EMANICS network.

To this end, three tasks were identified in Work-Package 6 :

- T6.1: Open source observatory: This task establishes a map of open source software dedicated to network and service management.
- T6.2: Open source repository: This task aims at a classification of high quality open source software developed within MAGIX and its partners.
- T6.3: Software development coordination: This task follows and contributes to a set of sponsored extensions of identified Open source software. Special focus is made on packaging, porting and integration, which are all critical factors to wide adoption.

Two of them were launched after the general assembly in January 2006. These are the Open source observatory task (T6.1) and the Software development coordination task (T6.3).

The design and development of an online observatory was undertaken in task 6.1 in coordination with the simple-web infrastructure. It is described in section 2.

For task 6.3, an open call[3] for Open Source software initatives support was issued to the participants on February 28th, 2006 with closing on March 15th, 2006. Out of the 6 proposals, 4 were selected on March 24th, in year one by the work-package leader together with the executive committee. These initatives are :

- ENSUITE Netconf Extension (INRIA + IUB, 15KEuros support)
- SCLI/CFEngine Integration (HIO+IUB, 15KEuros support)
- OSIMIS Linux Distribution (UniS, 7500 Euros support)
- Ponder Federations and Policy Language Extension (IC, 7500 Euros support)

[3] The forms to be filled by applicants to this Open Call is provided in appendix A of this deliverable.

These four software packages, the extensions planed under the EMANICS support, the distribution schemes, expected impact together with an interim report of the progress in the developments are described in section 3.

A conclusion on the first 10 months of operation of work-package 6 is given in section 4.

2 An online catalogue of available software for network management

Create an online catalogue of available software for network and service management arose before the actual start of the project to give users both an easily accessible tool with a user friendly interface and flexible search mechanism to a database of such software in one place – the EMANICS project public Web-site. The initial proposal was to design and implement a map of open-source management software as one tool (activity) and to offer additional visibility to selected software supported by the EMANICS project in a second task. One of the main aims was also to build an improved and best adapted version of existing "online software catalogues".

Various software inventories and lists publicly available in the Internet were evauated and considered to be a base for the software catalogue.

The following examples were found to be the most adequate for our domain and approach :

- the Simple Web "SNMP / Network Management Software" inventory http://www.simpleweb.org/software/select_obj.php
- a list of open source software available at http://wwwhome.cs.utwente.nl/~fiorezet/ron/Deliverable1.3.1.pdf
- the tool collection at IST MOME website

  http://www.ist-mome.org/database/MeasurementTools/?cmd=toolscategories

Although the above examples together include impressive and huge lists of management tools we have found that they have also had completely different structure of their content and some other flaws (e.g. some links were not up-to-date, some software tools didn't have information about their versions and supported platforms, many tools had only very few descriptive information available, …). During our investigation, we have also found one of the most important disadvantages of the majority of such catalogues – the lack of possibility of editing and updating the existing entries by end-users.

Taking all above arguments into consideration we decided create our own online catalogue of management software which would be free of mentioned flaws and filled with an updated and corrected content. Our catalogue should be also integrated together with the project official web pages and available via the web interface with an advanced search options. Finally to ensure long term survivability of the effort, we decided together with the Simple-Web team to make our databases compatible and maintain their consistency over time.

To fulfil such requirements we have searched and considered many different possibilities of implementation architectures and tools, for example Wikipedia-compatible solutions.

A Wikipedia-like approach seemed to be a basic framework, but unfortunately it is hard in defining data structure/scheme (a kind of web-forms) which can be filled by every partner with detailed description. Also later changes in the structure of Wikipedia are hard to perform (all descriptions have to be changed). It would be also good for our pur-

poses to have two interfaces available: one as read-only for widely open access and second for editing for every partner or for selected and registered users only.

For these reasons we decided to test various modules (components) for the Joomla CMS web-environment which can help to handle with databases and web-form interfaces. Additional benefit of using such Joomla's component is an easy and complete integration with the EMANICS official web pages created also in the Joomla CMS. We have tested several open-source modules of this kind (e.g. MosForms, DBQ) but only "DBQ Manager" was successfully installed and worked properly. Another big problem encountered by us, which is much common to open-source software is the lack of good documentation for the version of tested software and sometimes even the lack of any documentation or tutorial for such type of software. It makes the installation and implementation process of such software much longer and forced us to use a trial and error method during the tests.

Despite many problems encountered while using a free open-source version (1.4) of the "DBQ Manager" component for Joomla CMS, finally also by modifying this component's source code we have achieved all expected milestones shortly described below:

- " EMANICS Repository of Network Management Software" consists of the MySQL database with repository content and configuration tables, the advanced search mechanism with the web-interface accessible to all users (public), the "Add software" functionality accessible for registered users only and the functionality which allows the registered users with special privileges only to edit/update the existing content.
- The table of selected software listed in the EMANICS public repository and evaluated by the SimpleWeb.org and the EMANICS project participants as the best software packages.
- " EMANICS Inventory" consists of the open source software packages selected form the public repository and supported by this project.

All three modules mentioned above are fully integrated with the official EMANICS website and accessible via the "Software" item in the main menu (direct link: http://EMANICS.org/component/option,com_dbquery/Itemid,93/). Their main functionality and available options as well as the short explanations of how to use them are described below together with the screenshots given for better understanding and visualisation purposes.

After successful login to EMANICS pages and choosing the "Software" in the main menu a user can see for example such a page:

Figure 2.1 The main page of the " EMANICS Repository of Network Management Software".

This main page of the "Software" menu includes three different parts (as shown above):

- the welcome text for EMANICS repository and some hints which shortly explain to the users their accessibility to the various functionality of this repository,
- the table of software packages selected from the EMANICS public repository and recognized as the best and commonly used management tools (see below - Figure 2.2),
- the direct link with icon to the search form of public repository.

The best software packages listed in the Emanics public repository are selected for you below

Name	Source	Description	More
Cacti	Ian Berry	RRD Front-end	More
Ethereal	Gerald Combs	Ethereal is a network protocol analyzer for Unix.	More
libsmi	TU Braunschweig	A library to access SMI MIB information	More
MRTG	T. Oetiker and D. Rand	A tool to monitor the traffic load on network links.	More
net-snmp	Univ of California, Davis	Various tools relating to the Simple Network Management Protocol	More
ntop	Luca Deri	A tool that shows the network usage, similar to what the popular 'top' Unix command does.	More

Figure 2.2 The table with the best software packages.

In order to find a specific software package(s) there is an advanced searching mechanism implemented and accessible for all users by choosing the “Search repository” submenu item or via the direct link with a magnifying glass icon. It uses a web-form interface to get input data from a user and looks like an example screenshot placed below.

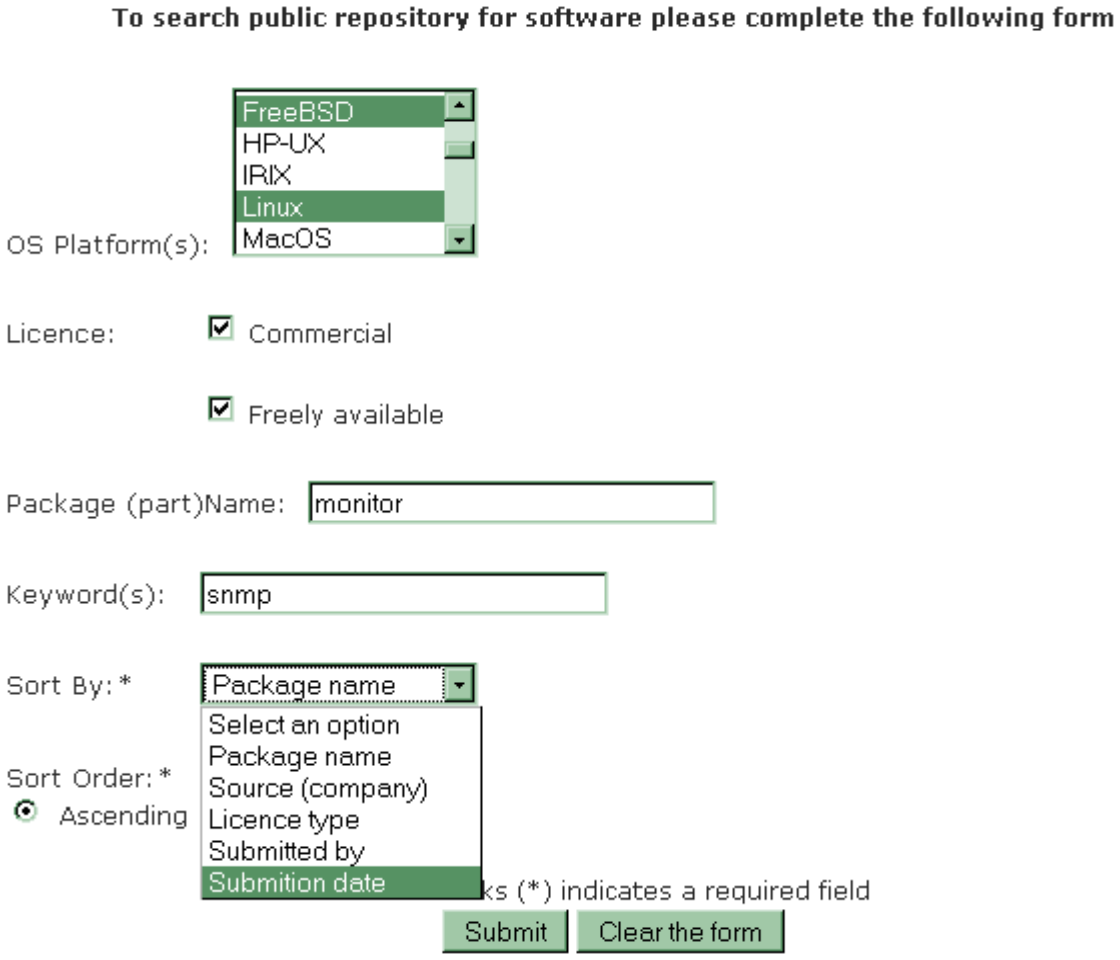

Figure 2.3 The web-form interface for searching software inside the repository

The results of search functionality are presented as a table on a single web page or several pages with page-navigation links on the top and with sortable columns (by clicking column name). The column called “Name” displays a software package name as an HTTP URL to the product or its vendor home page, if available. Users can also change the number of rows displayed at one web page by changing the “Display” parameter value (HTML form list). Part of the first page of the results for searching for all software included in the repository is shown below as an example.

Results matching your criteria for search Emanics public repository are presented below

<< Start < Prev 1 2 3 4 5 6 7 8 9 10 Next > End >>

Results 1 - 10 of 222

Display : 10

Name	Source	Description	More
3Com Network Supervisor	3Com	Network Management Software	More
3Com Network Supervisor Advanced Package	3Com	Works with 3Com? Network Supervisor Version 3.5 to increase the number of devices you can manage and lets you easily manage agents on those devices.	More
AdRem Free Remote Console	AdRem Software	Free remote access to NetWare server console	More
AdRem NetCrunch	AdRem Software	Network management, network monitoring, diagnosing and reporting tool	More
AdRem Server Manager	AdRem Software	NetWare server performance and connection monitoring (CPU, memory, network performance); user monitoring (e.g. opened files, requests per second);file/disk/directory/trustee/NLM management; multi-server configuration and software distribution.	More
AdRem sfConsole	AdRem Software	Encrypted, Web and eDirectory-enabled remote/local access to the NetWare console; access rights management; emergency connection and file transfer; user activity audit; ability to open multiple server and program screens; low footprint; single sign-on.	More
Advanced SNMP Agent and MIB-Compiler	DMH Software	Advanced SNMP agent,MIB compiler and other portable internetworking software components.	More
AdventNet Agent Toolkit C Edition	AdventNet, Inc.	AdventNet Agent Toolkit C Edition is a rapid prototyping and development tool for building SNMP, TL1, and CLI agents in strict ANSI C language. In addition to this it also enables the development of Multi-protocol agent with SNMP, TL1, CLI, and HTTP protocol support. The run-time agent developed using AdventNet Agent Toolkit C Edition is very modular,	More

Figure 2.4 Part of the results for searching repository functionality.

To see more information about a particular software presented in the general results table (see above), users can click on the package name which is the URL to the product webpage or on the “More” button located on the right side of this software description.

When the second choice is made, by the user, the page with the detailed information about selected software from the EMANICS public repository is presented. It looks analogically to the following example screenshot (the “Cacti” details page).

ID	369
Package name	Cacti
Version	0.8.6h
Source	Ian Berry
Licence	free
OS platforms	HP-UX, Linux, Solaris, SunOS, Unix, Win98, WinNT, Win2000, WinXP
Web site	http://www.cacti.net/
Short description	RRD Front-end
Keywords	Cacti, RRD, MRTG,
Submitted by	Aiko
Submitter's e-mail	pras@cs.utwente.nl
Submision date	2005-04-07 15:18:32
Last change by	rszuman
Last change date	2006-08-10 15:27:42
Long description	Cacti is a complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. Cacti provides a fast poller, advanced graph templating, multiple data acquisition methods, and user management features out of the box. All of this is wrapped in an intuitive, easy to use interface that makes sense for LAN-sized installations up to complex networks with hundreds of devices.

EDIT

Figure 2.5 The detailed information page about selected software (“Cacti”) after choosing the “More” button for it.

All the information available on the details page of any software package can be edited and updated by users with a user-friendly web interface. This functionality is available only for registered users with special privileges in Joomla (e.g. editor rights). When the user with appropriate privileges to update this information is logged in then the "EDIT" button is visible on the bottom of the details page (see example above). To edit the details of selected package, the user has to complete the special form (Pict.2.6).

Figure 2.6 The web-form interface for editing/updating detailed information of selected package ("SCLI").

The last but not the least of the modules mentioned on the beginning of this chapter is the " EMANICS Inventory", which includes open-source software packages supported by the EMANICS project and developed in the scope of it. Current view of this inventory first page is presented below.

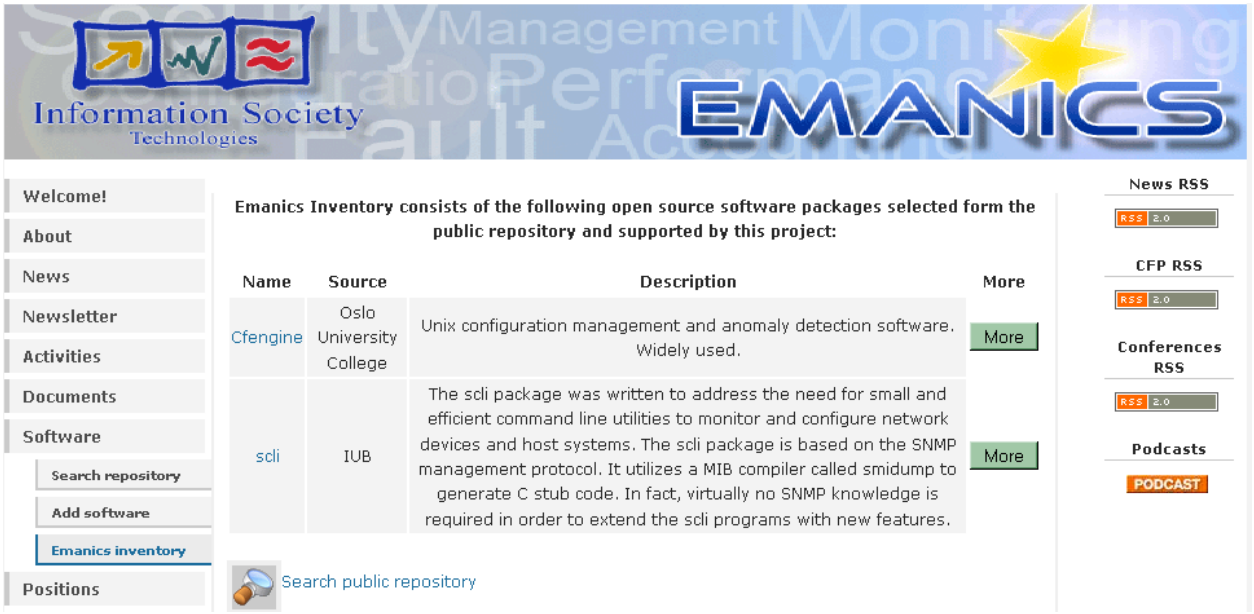

Figure 2.7 The " EMANICS Inventory" which consists of the software packages officially supported by this project and already included/added in the public repository.

Finally it should be mentioned that the content of the database table with software descriptions used by the inventory is based at the SQL-compatible export of the SimpleWeb.org database table. Content of this table was corrected and updated after importing it into the EMANICS repository database, while structure of the table was extended to provide better functionality. The updated and coherent database was then exported from the EMANICS repository and sent back to the SimpleWeb for synchronisation purpose. This cooperation and coordination with the Simepl-web will continue for the duration of the network.

3 Supported Open Source Software

3.1 ENSUITE Extensions

3.1.1 ENSUITE : a Netconf platform

EnSuite is a LGPL implementation of the Netconf configuration protocol in its SSH version. It consists of YencaP Manager which is both a web server and a Netconf manager, and YencaP which is the EnSuite implementation of a Netconf agent. EnSuite is fully developed in pure Python programming language. Documentations, software releases, bugs, forum, news are freely available in the web (http://libresource.inria.fr/projects/ensuite) and integrated in the Libresource environment that is a cooperative platform for software development (like sourceforge).

EnSuite proposes two original capabilities that are typically advertised at session startup. The first one is an access control system, that allows to filter Netconf operations according to the privileges specified in the local agent policy. The second one is a MIB module deployement and management capability which works somewhat like OSGI bundles. In particular, it allows to deploy new extension modules as zip archives into the agent, install and load them dynamically to the program at execution time. This is very useful to update the YencaP software transparently or to deploy new Netconf services at large scale.

EnSuite uses many technologies and standards among XML, XPath, XSLT, DOM, HTML, javascript, CSS, HTTPS, SSH, some of which being dedicated to Netconf, and some others to the web components. EnSuite is very modular, extensible, and well structured thanks to the application of Design Patterns.

The distribution, in terms of lines of code, is as follows:

- Agent: 24 000
  - Core: 11000
  - Modules: 13000 (of which 6000 are from the BGP module and 7 are from the RBAC module)
- Manager: 10 600
  - Core: 9200
  - Modules: 1400

EnSuite is fully compliant with Linux distributions, since it is distributed as:

- rpm package for Fedora Core,
- deb package for Debian and,
- tar.gz for all Linux distributions.

It also supports the classical lifecycle management of Linux services:

- /etc/init.d/yencap [start|stop|restart]
- /etc/init.d/yencap-manager [start|stop|restart]

The dependencies in terms of software packages are as follows:

- python-4Suite-XML-1.0-rc4
- python-amara-1.1.7
- python-paramiko-1.6.1-1
- PyXML-0.8.4

One additional package is required for the agent, quagga-0.98.6-1, and one for the manager, pyOpenSSL.

3.1.2 The Yenca agent toolkit

General architecture

YencaPs' design follows the layered architecture of the Netconf configuration protocol. It intends to be as modular as possible to allow people to add new capabilities, new operations and extend the data model. To provide this level of modularity, it makes use of

some well-known Design Patterns: Command, Singleton, Facade, Composite. YencaP is extensible in three of the four layers of Netconf:

- Server which is the transport protocol layer and can be extended to SSH, BEEP or SOAP or other experimental protocol,
- Operation (or Command) which is extended to Get_Config, Edit_Config, Lock_Config and so on,
- Module which is the Content layer and can be extended to provide new management data.

The major advantage of this flexible architecture is that it allows to add new modules or operations without any change to the code of YencaP core stack. This is not always true but in a majority of case, it is, in particular for modules.

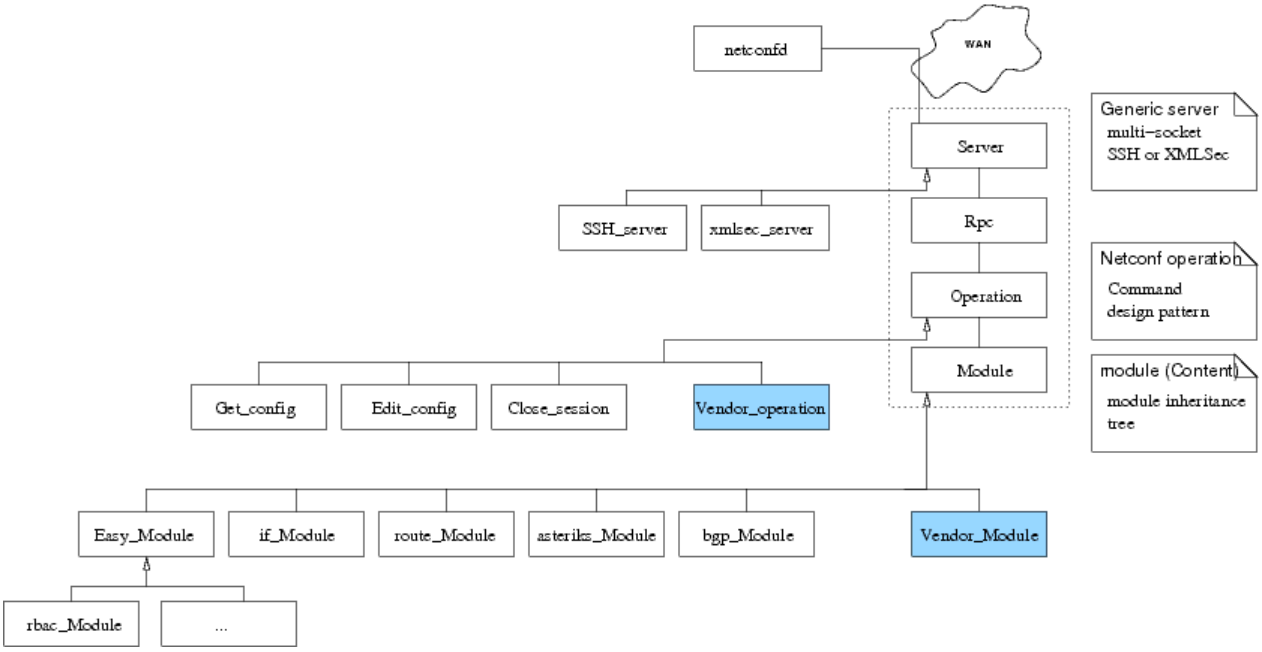

Figure 3.1 : The yenca tree

YencaP configuration cache

YencaP uses a cache mostly because of the xpath capability. When receiving an XPath request, it is sometimes hard to find the concerned modules because of relative expressions and also the axes which can be very complex (ancestor, childs, and so on). So a cache is maintained according to a cache lifetime specific to each module. A cache makes it very easy to apply XPath requests to the configuration tree. While it consumes more memory, it allows to improve the response time.

Layers interfaces

YencaP uses well-defined interfaces to communicate between layers. For example, YencaP defines a ModuleReply class that allows each module to send a standard reply to the Operation layer. The reply can be an error, an positive reply (ok) or configuration data.

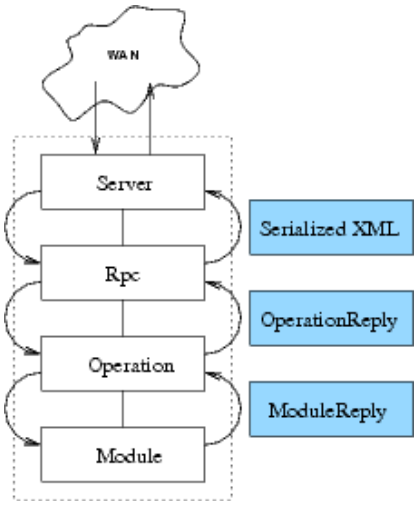

Figure 3.2 : Layers in the Yenca framework

Operations

Netconf operations are implemented as a Command Design Pattern, which consequently makes the set of operations extensible without changes to the core of YencaP. YencaP has a file descriptor for operations which allows their dynamic instantiation from their names and with a reflection process like in many modern languages. This means that it is possible to implement a new Netconf operation, provide its description and it will work without any change to the code of YencaP.

Existing modules

A set of modules is natively included into YencaP. They all inherit from a generic Module class and therefore follow the same API. Consequently, they are all called in the same way and this makes the code of YencaP core identical whatever the module. As for operations, a module can be added without any change to the code. This independance is a major advantage since module developers can benefit from the new versions of YencaP core without problem.

YencaP delegates the management of the different datastores to modules because the way to do it is too dependant on the managed platform (CLI interface, XML data file, Unix pipe, ...). Not all modules support all Netconf operations. For example, IPsec only supports get-config. Here is the list of implemented modules:

- Asterisk Module
- Network Interfaces module
- RBAC module
- Route module
- BGP Module
- RIP module

Each module defines its own XML data model. This XML subtree is plugged into the Netconf agent XML configuration, which can be seen as a collection of XML subtrees. A XML subtree can be plugged at any level in the XML hierarchy. YencaP defines to subclasses of Module, namely Easy_Module and CLI_Module. They make its easier to im-

plement new modules by providing a set of methods, for example to connect to the CLI via telnet and send commands or also to autogenerate an XSL file from an edit-config request.

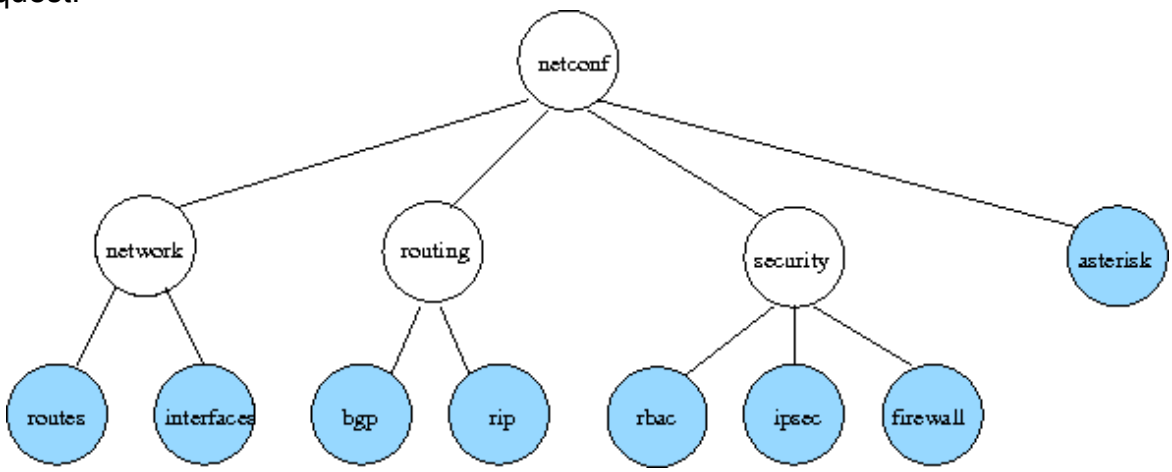

Figure 3.3: YENCA available modules

3.1.3 A Netconf manager (YencaP Manager)

YencaPManager is both the Web server and the Netconf manager (client). It implements 95% of the Netconf draft for the manager side. It has basically two main web interfaces: one dedicated to agent selection in case the number of agents is huge, and one dedicated to the management of one agent with all the Netconf operations and the supported modules.

General architecture (Composite window)

Pages hierarchy and composition

A HTML page can be of different types, depending on the navigation context. A page as defined in YencaPManager can be serialized to a string in order to be sent to the web browser of the human manager. We defined three types of Pages:

- MainPage: when a manager is selecting the devices to manage
- AgentConnectedPage: when a manager is connected to (and is managing) a device with Netconf
- AgentReachablePage: when an agent is reachable but is not connected.

AgentConnectedPage itself can be of different types, but they have shared components like for instance, the menu of a connected agent. Therefore the menu is made in the AgentConnectedPage. Here is the list of such pages:

- XpathPage: for sending get-config request based on XPath,
- SubtreePage: for sending get-config request based on subtree filtering,
- LockUnlockPage: for locking and unlocking Netconf devices,
- EditPage: for sending edit-config requests to the Netconf devices,
- and so on… with all Netconf operations.

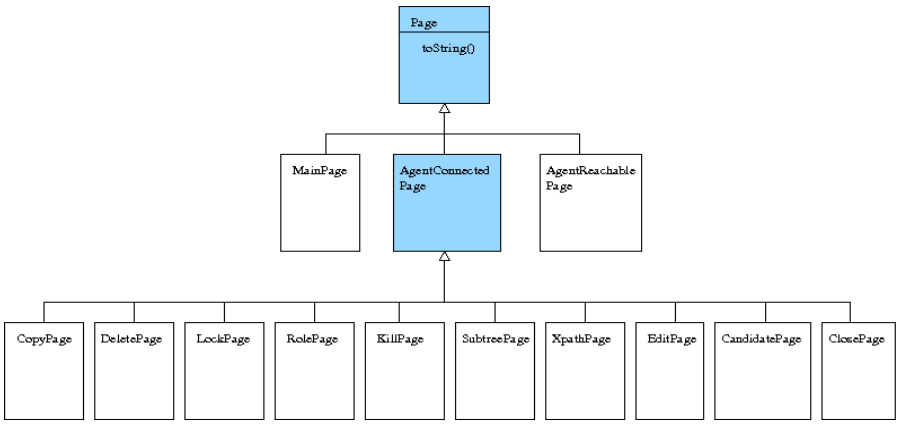

Figure 3.4 : Page hierarchy on the YENCA Manager

Like all HTML pages, a Page is basically a composition of components: a header, a content (menu and real content), and a footer. All of them can be in turn compositions of other smaller components, and so on. This is an application of the Composite Design Pattern. The page rendering is managed by a CSS file.

Agent filtering

The main page of YencaPManager allows to select Netconf agents based on multiple criteria:

- IP address or network
- State (unreachable, reachable, connected)
- Group (server, router, ...)
- Supported capabilities

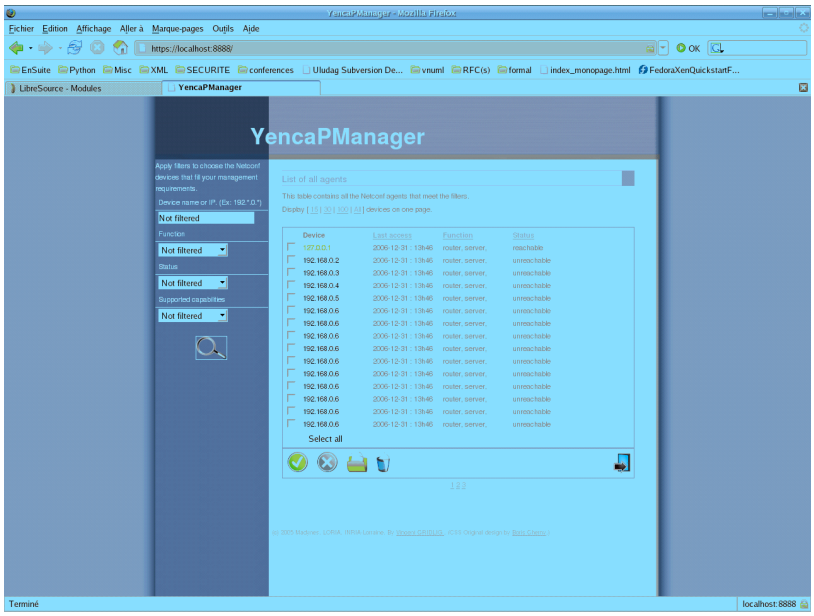

Figure 3.5: A sampl YENCA Manager page

The agent state is maintained by a separate thread which pings agents periodically. Using a thread improves the response time of the application.

Netconf Agent page

The Netconf agent page provides two important things: the module forms that allow to update the configuration values and the standard Netconf operations forms. Modules inherit from the Module class. Each module has to provide an XSL file that renders the XML data into HTML code.

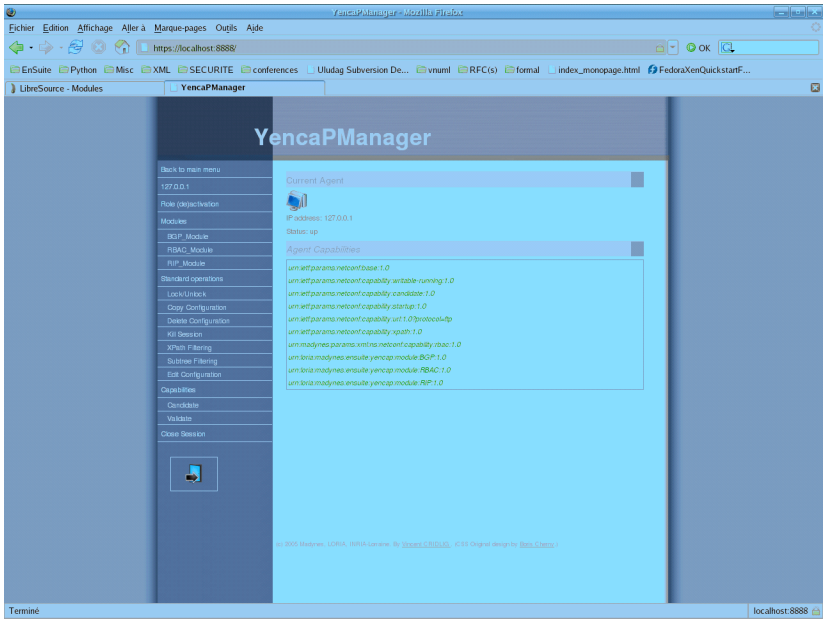

Figure 3.6: A samepl YENCA Agent management page

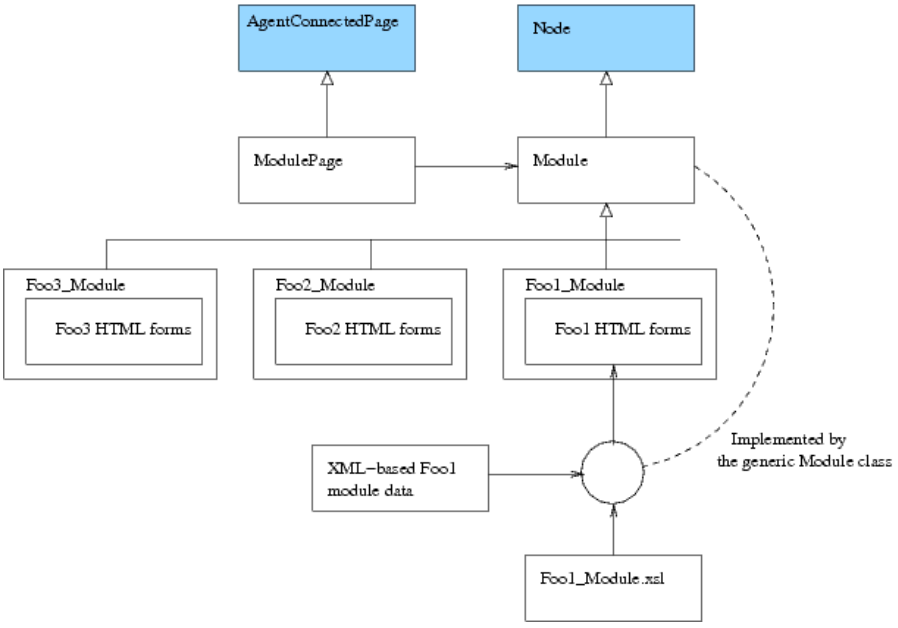

Figure 3.7: Agent page structure

3.1.4 Detailed list of extensions planed under the support of EMANICS

The following three efforts are planed :

1. **Ensuite core : maintenance and distribution**

This activity aims at ensuring maintenance of the core distribution of the Ensuite platform and at providing support for its distribution.

2. **Ensuite extension for AAA functionality**

   The current security mechanisms of Ensuite are based on the IETF endorsed SSH or the INRIA-Madynes developed RBAC mechanisms. Currently, IETF has a pending work item to define RADIUS attributes for network management authentication and authorization. We will contribute to this work by adding a Radius/Diameter based solution to ENSUITE in order to address the management of large networks. Both IUB and INRIA-Madynes have developed RADIUS-based authentication and authorization solutions for specific networking/management systems and will build on their experience to provide this support to ENSUITE.

3. **Ensuite extension for notifications.**

   Currently no notification support exists in Ensuite. Support of notifications in Netconf is currently under investigation in the IETF Netconf working group. Depending on the evolution of this standardization activity, support for notifications will be implemented in ENSUITE.

3.1.5 Expected impact

NetConf is a new protocol and it remains to be seen how successful it will be in practice. As such, there are many possible long-term scenarios, ranging from NetConf being not adopted as a management protocol (in which case the work would become irrelevant) up to NetConf being highly successful and finally replacing SNMP, in which case the work would be extremly timely and would put EMANICS in the fore front of the technology experts in this are. In other words, there is some risk involved (which is not under control of EMANICS).

EMANICS can clearly benefit by leading implementation efforts in this space and thus gaining influence on the relevant IETF processes (where implementation experience still is considered very important). By improving the NetConf prototype and extending it with NetConf extensions under discussion in the IETF, it will be possible to influence and actively contribute to specifications and standards.

3.1.6 Progress report

Below we provide the progress report in form of a changelog since the beginning of the EMANICS support campaign. The report differenciates the agent (YencaP) and the manager. In these first months, we focused our work mainly on the evolution of the core platform and on its maintenance (task 1).

3.1.6.1 YENCAP

Release 2.1.7

- Access control is done also for copy-config and delete-config
- API changed for Module class
  - def get(self, configName):
- def getConfig(self, configName): where configName is one of "running", "candidate" or "startup". (These constants are accessible via C.RUNNING, C.CANDIDATE, C.STARTUP).
- def copyConfig(self, sourceName, targetName, sourceNode = None):
- def validate(self, targetname, moduleNode = None): where targetname is one of

"running", "candidate", "startup" or "config" (the last case requires that the moduleNode must be given as parameter).

- Support validate operation from Maximilian Huetter.

Release 2.1.4

- RBAC improvement for being able to activate a junior role of the assigned roles.
- 4Suite rc4 solved the XPointer bug of the BGP module. Another bug was fixed in BGP module
- Small bug fix in copy-config and edit-config due to namespaces.
- Operations are now loaded by the OperationFactory. Operation information are read from operations.xml by OperationReader. OperationReader now uses amara. OperationFactory now uses the same system as ModuleFactory. OperationManager is the central point for the Operation management.
- Edit-config has been improved. The default operation is first set to the default value. Then, for each module, it checks that the operation attribute is present or not in the parent nodes because this changes the default operation (merge, replace, create, delete). This happens before calling the modules.
- Access control for edit-config has been improved. It selected the authorized nodes in the received request (using XPath scopes). Then if an operation attribute is in the parent nodes then the operation is not allowed since the permissions are granted only on the subtree, not the parents.

Release 2.1.3

- a lot of cleaning (unnecessary imports, XMLSec support)
- continue migration to amara
- lookup tools to build the RPM

Release 2.1.2

- YencaP is now able to upload new modules, install them and load them, all this dynamically.
- split ModuleManager into:
- ModuleManager which is responsible for (un)loading modules on demand. This is implemented as a Singleton,
- ModuleFactory which is responsible for building modules on demand. This is implemented with both a Singleton and Factory design Pattern,
- ModuleReader (formerly ModuleParser) which is responsible for parsing modules on demand (with amara). This is implemented as a Singleton design Pattern.
- Improvement of DatastoreManager code.

Release 2.1.0

- IANA port for Netconf over SSH is now set to 830.

Release 1.1.32

- you don't need to install yapps2, neither to setup the YAPPS environnement variable. We included the two following files to our distrib : yapps2.py and yappsrt.py. This is mostly because yapps2 is not yet available in all the standard

Linux distributions as a package. We will consider to change that whenever it is available as an RPM (the deb package is already available).
- XPath request "/" now replies with the full document (if the user has sufficient privileges),
- LogModule now calls the super constructor of EasyModule, not Module,
- The constructor of the modules has been changed and now, each module must call the constructor of the super class Module or EasyModule,
- ...

Release 1.1.25

- Access control is now developed for edit-config. A wiki page is under work to explain the way it works (See YencaP configuration guide),
- Environment variables (YENCAP_HOME...) are no longer required,
- /usr/local/Ensuite/yencap is changed to /usr/local/ensuite/yencap, in order to follow the Linux "best practices",
- /etc/Ensuite/yencap is changed to /etc/ensuite/yencap, in order to follow the Linux "best practices".

Release 1.1.24

- We now provide rpm, deb and tar.gz package.

Release 1.1.23

- Candidate capability is now fully implemented: <commit> and <discard-changes> were missing
- Update <copy-config> according to the new internal design
- Fixes a lot of minor bugs

Release 1.1.22

- fixes a set of bugs in subtree and XPath filtering
- adds support for XML namespaces per module
- Xpath filtering allows the use of prefixes. It is the recommended way.
- moduleResolver has been splitted into two classes: DatastoreManager and ModuleManager.
- DatastoreManager class is responsible for managing the different datastores. In particular, it stores a copy of the running configuration.
- ModuleManager class is responsible for storing the list of modules. It can load, unload and reload modules properly.
- the syntax of the modules.xml file has changed to allow namespace and cache life-time definition

3.1.6.2 YencaPManager

Release 2.1.5

- The page generation process of YencaP Manager was refactored. All pages now inherit from a common model (Page class). This class has three sub classes: mainPage for the agent selection, AgentConnectedPage for managing a given agent and AgentReachablePage which gives minimal info about an agent. AgentConnectedPage has many sub classes corresponding to different things:
  - Forms for Netconf operations
  - Module forms using XSLT

It should be now much easier to build new pages. We also started to update the documentation of YencaP Manager.

Release 2.1.4

- BGP module for yencap Manager side. It allows to define neighbors, route-maps and assignements between them.

Release 2.1.2

- Add support for hot-pluggable modules
- Foo_Module.zip is a sample module that can be used for hotplug demonstration.

Release 2.1.1

- The GUI changed a lot to allow efficient Netconf agent filtering. In particular, in the context of hundreds agents, the GUI makes it possible to select a subset of the agents depending on their status, function, capabilities and IP address.
- IANA port for Netconf over SSH is now set.
- Device status (reachable and unreachable) is now maintained in a separate thread. This improves response time of the application when it must check status of a lot of Netconf agents.
- YencaP Manager now migrates slowly towards python-amara which is really simple for parsing XML files. *python-amara* is a new package requirement which is distributed in standard FC5 distribution.

Release 2.0.17

- In the web form concerning Subtree and XPath filtering, the prefix list is set automatically accroding to the modules.xml file,

Release 2.0.8

- /usr/local/Ensuite/yencapManager is changed to /usr/local/ensuite/yencap-manager, in order to follow the Linux "best practices".
- /etc/Ensuite/yencapManager is changed to /etc/ensuite/yencap-manager, in order to follow the Linux "best practices".
- Namespaces bugs in RBAC Netconf requests are fixed.

Release 2.0.7

- Add <commit> and <discard-changes> features

3.2 SCLI/CFEngine integration

The objective of this work item is to marry two open source software packages. The first one is cfengine, a policy host configuration system [22]. The second one is scli, a command line interface to network devices running on top of SNMP [23]. In this section, we will describe each package in turn and then outline the work we intend to do, describe the expected impact, and document the progress that has been made so far.

3.2.1 Software description

3.2.1.1 cfengine

Cfengine is a configuration management environment that includes a high level policy language and a low level logic engine. Cfengine ties configuration policy to system monitoring, thereby allowing feedback and reactive configuration. Cfengine does not currently interface with SNMP.

Cfengine can be used as a scripting language for autonomic maintenance of Unix-like computers. For large systems with many different flavours of operating system, what is needed is a disciplined way of making changes which is robust against reinstallation. The idea behind cfengine is to focus upon a few key areas of basic system administration and provide a language which removes decision clutter from scripts, by providing a declarative language, so that the transparency of a configuration program is optimal and management and implementation are kept separate.

Cfengine focusses on a few key functions which are handled rather poorly from scripts. It eliminates the need for lots of tests by allowing you to organize your network according to classes. From a single configuration file (or set of files) you specify, using classes, how your network should be configured -- and cfengine will then parse your file and carry out the instructions, warning or fixing errors as it goes. The basic functions include:

- Checking and configuring the network interface
- Editing textfiles
- Maintaining symbolic links, including multiple links from a single command
- Checking and setting the permissions and ownership of files
- Tidying junk files which clutter the system
- Systematic, automated mounting of filesystems (Unix)
- Checking for the presence of important files and filesystems
- Controlled execution of user scripts and shell commands.

Another component of cfengine performs machine-learning functions. There is no system available in the world today which can claim to detect and classify the functioning state of a computer system. Cfengine incorporates a framework, based on the current state of knowledge, for continuing research into this issue. In version 2.x of cfengine, an extra daemon cfenvd is used to collect statistical data about the recent history of each host (approximately the past two months), and classify it in a way that can be utilized by the cfengine agent.

The current cfengine learning abilities do not extend to SNMP MIBs. By interfacing with scli, it will be possible to incorporate reactive management (so called Event Condition Action behaviour) based on SNMP devices. Communication between cfengine and scli can also allow eventual control of the devices through the cfengine language interface.

3.2.1.2 scli

The program called scli provides an efficient to use command line interface to display, modify and monitor data retrieved from SNMP agents. It runs on simple ASCII terminals and does not require any graphical user interface capabilities. The tool provides command line editing, completion and history capabilities to make it easy for network opera-

tors and system administrators to use this tool, even if they cannot remember the precise command syntax.

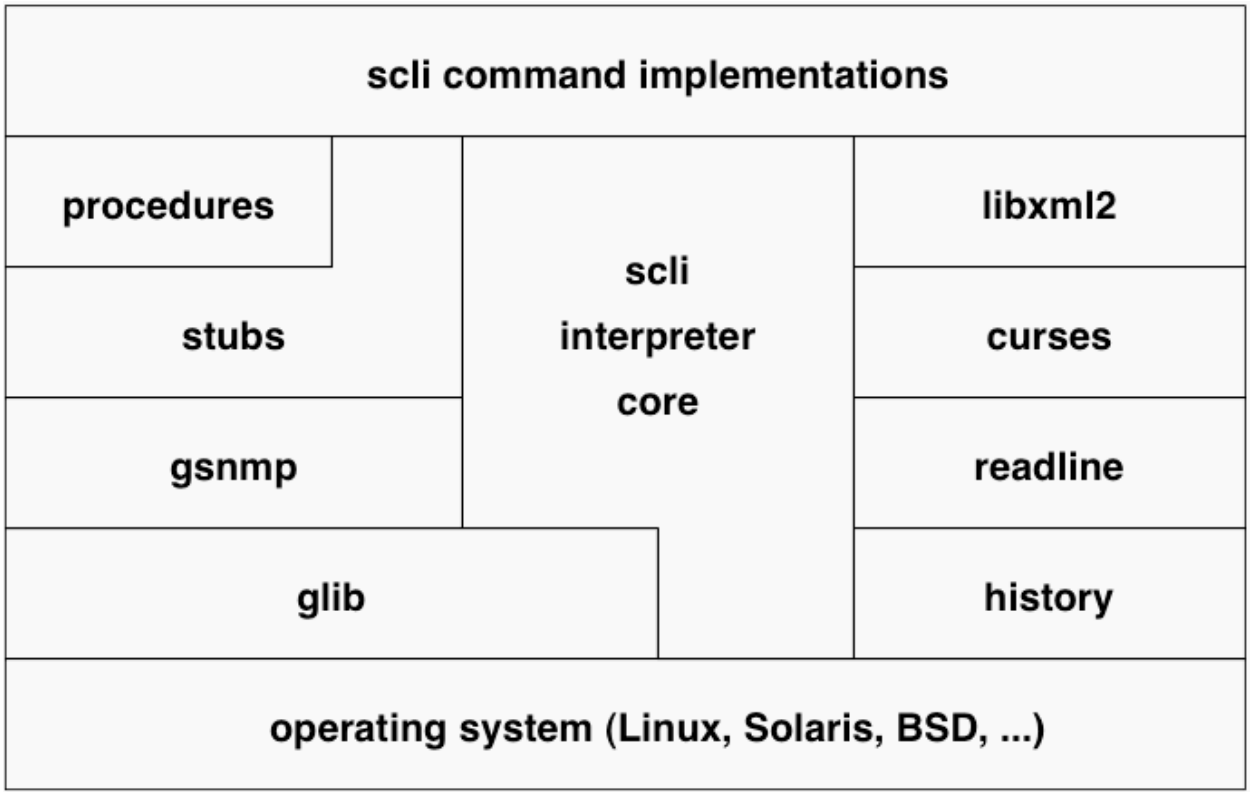

Figure 3.8: scli architecture

The overall software architecture is shown in the Figure above. The package uses the glib library to achieve portability and to reuse generic data structures such as lists and dynamic strings. The SNMP engine gsnmp has been derived from the gxsnmp package and was subsequently modified to fix bugs and to improve stability. The SNMP engine itself uses the glib.

The gsnmp library provides roughly the same low-level functionality as many other SNMP APIs. Since it was felt necessary to hide programmers from the low-level SNMP programming details, it was decided to use a MIB compiler to generate C stubs from MIB modules. The stubs consist primarily of C structures which represent MIB table rows or groups of scalars plus a set of stub functions which can be used to read/write these structures. The implementations of the stub functions serialize/deserialize the C structures into SNMP varbind lists. They also validate the data to ensure that the elements of the varbind lists have appropriate types and sizes.

The scli command implementations either use the stubs directly or they use so called MIB procedures. MIB procedures extend the stub interface with specific functions for common operations like creating rows in certain MIB tables or iterating over certain MIB tables. MIB procedures are by definition MIB specific and implemented entirely by using the stub interface.

The scli interpreter core provides all the infrastructure needed to register commands, to tokenize the input stream, to locate and execute the function implementing a recognized command and to finally display the results on stdout or via a pager. The interpreter uses

the GNU readline and history libraries for command line editing and the curses library for screen management. It also uses glib data types internally. All state information is bound to the scli interpreter.

The interpreter core and some command implementations also use the libxml2 library to create and manipulate XML documents. By using a dedicated XML library, it is possible to ensure that any generated XML output is well-formed.

3.2.2 Detailed list of extensions planed under the support of EMANICS

The main goal is to integrate SNMP access to network devices into cfengine and cfenvd. The approach taken is to use the gsnmp engine and the compiler generated stubs plus a semantic layer which is shared between scli and cfengine. In the end, cfengine should be able to monitor, control and configure remote devices such as layer two switches via SNMP.

Including SNMP data into cfengine will allow monitoring and automated trend analysis of SNMP enabled devices. In the future, it is likely that many, if not all, network devices will run an operating system like Unix/Linux on a small blade card. At such a time, cfengine will be able to run directly on these devices. Today, few systems have this capability, but a remote Linux/Unix machine will be able to query and even configure the devices to a degree, allowing the intelligence to reside in a close-by PC.

3.2.2.1 Monitoring of Parameters using SNMP

The cfengine includes a daemon cfenvd which is collecting longer term statistics in order to detect anomalies. The statistics typically originate from the local host. The goal is to extend this daemon so that it can also collect key parameters from remote systems using SNMP. The approach will be to define an interface over which additional data sources can be integrated with cfenvd. The scli program will be extended to provide such a data source.

Data are read into the daemon and used to update a database using a special algorithm that uses a constant memory size. The daemon automatically adapts to the changing conditions, but has a built-in inertia which prevents anomalous signals from being given too much credence. Persistent changes will gradually change the `normal state' of the host over an interval of a few weeks. Unlike some systems, cfengine's training period never ends. It regards normal behaviour as a relative concept, which has more to do with local stability than global constancy.

The final size of the database is approximately 4MB. Measurements are taken every five minutes (approximately). This interval is based on auto-correlation times measured for networked hosts in practice.

Cfenvd analyses the arrivals and compares them to the learned history. It then sets a number of classes or conditional boolean variables in cfengine which describe the current state of the host in relation to its recent history. The classes describe whether a parameter is above or below its average value, and how far from the average the current

value is, in units of the standard-deviation. This can be plotted and analysed further using tools provided by cfengine.

3.2.2.2 Policy Enforcement via SNMP

The cfagent component of cfengine enforces policies written in the cfengine language on the local host system. The goal is to enhance the cfengine language in such a way that it is possible to define policies that should be enforced on network elements that can talk SNMP.

It was agreed to focus on the configuration of VLANs in order to demonstrate the interworking of these software packages. Cfengine implements changes using a method of convergent operations. In recent work, HIO has shown how to implement such operations in a service framework. Here we shall use that framework to implement operations in conjunction with scli. A convergent operator has the property that its repeated application will eventually lead to the base state, and no further activity will be registered thereafter. This requires a slight modification of the operators since the base state must be checked for explicitly.

The implementation makes use of promise theory. The benefits of promise theory are two-fold. Autonomy of the nodes in a promise graph forces us to make all relationships and policy atoms explicit, and the service-nature of the promises makes all changes appear on an equal footing, no matter whether the resources are local or remote, whether the communication is over an internal bus or an external network. This makes certain descriptions that we are used to taking for granted seem cumbersome in promise theory, but it also means that we avoid problems like hidden inconsistencies. Finally, it allows us to take any component of a system and make it into an independently autonomic device.

Using promises we can therefore implement a declarative language interface in cfengine for the SNMP controlled devices. This will require the implementation of a feedback loop through the interface.

3.2.2.3 Concrete Development Steps

The integration of cfengine and scli will be achieved by a loose coupling where a cfengine process may start scli processes in order to collect data or to enforce certain configuration policies. A simple protocol will have to be defined to support the communication between the components. We expect that this protocol will be executed over a local pipe.

Scli will be extended to support additional machine readable output format and a monitoring mode where scli pushes periodically collected statistics to cfenvd. Furthermore, the scli command set will be extended to provide semantics which allow the repeted execution of commands without causing failures (guarded commands).

The cfengine/cfenvd programs will be extended such that they can initiate a pipe to scli and send command and parse the returned output. The details of the protocol between scli and cfengine/cfenvd are still under discussion.

3.2.3 Expected impact

Cfengine as described is used by millions of users worldwide and has an established user base. Scli has been described in a LISA paper butso far is only used by a rather small number of sites. The integration of these two packages will (a) add important features to cfengine in order to configure for example VLAN memberships via cfengine and (b) enhance the technology on which scli is based and at the same time increase its visibility.

Cfengine is published under the GNU GPL2 license. Cfengine is regularly included in Linux and Unix package distributions. Scli is published under GPL, the underlying gsnmp library is under LGPL, and the libsmi package used to generate stub procedures is under a BSD-like license. Scli has been packaged for several Linux distributions including Debian GNU/Linux.

By integrating these two packages, we also integrate software coming from and used by slightly different communities, namely system administrators and network managers. This integration will hopefully open the floor for some synergies and ideally network managers will start to adopt more automated approaches to configuration management while system administrators will take advantage of very lightweight monitoring protocols.

3.2.4 Progress report

Scli has been updated and a new web page has been established for the software project. Work has been started to provide a simple HTTP like protocol to invoke commands on the scli interpreter. In addition, work has been done to support multiple scli interpreter instances in an single scli process.

In addition, work has started to add additional commands to scli to monitor and configure VLANS based on the Q-BRIDGE-MIB [RFC4363]. Some more design work is needed in order to design an orthogonal set of commands which have a semantic that suits the cfengine integration (guarded commands). HIO and IUB have discussed a possible syntax for the promise interface.

The cfengine code has been updated in version 2.1.21 to make it suitable for integration. Significant reworking of the code was performed to provide a general interface between cfengine and other sources of data, which will include scli. The data structures and methods which update them are now of a general extensible form suitabel for further testing and research using the VLAN example. The code has been tested for several weeks and is working correctly.

Additional short-term anomaly detection methods has also been added to the code based on "Leap Detection Testing". This will be useful for more rapid reaction to abnormal states in VLAN. We are currently testing this in Oslo.

We have planned a meeting at the manweek conference to coordinate progress. We expect the integration work to last until after the new year.

3.3 Ponder extensions, packaging and public availability

3.3.1 Software description

Ponder2 is a policy service aimed at the implementation of autonomous self-managed cells (SMC). A SMC is an autonomous collection of software and hardware components

such as those in a body-area network, a room or even a large-scale distributed application. Ponder2 comprises a policy interpreter for implementing obligation policies (in the form of event-condition-action rules) a domain-based service where managed objects can be explicitly grouped in hierarchical and overlapping domains and an internal event propagation mechanism for triggering obligation policies. Conceptually the software is derived from past experience with the development of the Ponder system however it has been entirely re-designed to a self-contained implementation that can scale from small embedded systems to larger distributed systems and networks. Because it is self-contained the implementation can be easily embedded into other services.

We have had considerable experience with the use of policies as a means of specifying adaptive behaviour in network management and other applications. The use of interpreted policies means they can be easily changed without shutting down or recoding components. The policy service maintains adapter objects for each of the components on which management actions can be performed. This includes the sensors and other devices present within a SMC, services within those devices and remote SMCs. These adapter objects (also called managed objects) are grouped in a domain structure that implements a hierarchical namespace e.g., similar to a file system. However, unlike in a file system, domains may overlap and a managed object may belong to several domains. Domains and policies are managed objects in their own right on which actions can be performed e.g., adding/removing an object from a domain, enabling or disabling a policy. Consequently, events can trigger obligation policies (ECA rules) that can enable or disable other policies and change domains and domain membership. In essence domains are a means of classifying and grouping the managed objects in a hierarchy and permit them to be addressed using simple path expressions.

When an obligation policy is created in the policy service, an event subscription for that event is sent to an internal event bus. Upon receiving a notification, the policy service evaluates all the obligation policies triggered by that event. Event adapters can also be created for receiving events from external event buses such as Elvin, Siena or XMLblaster.

The policy service has been implemented with particular focus on flexibility and the ability to load all the code needed on-demand. This enables us to use it across a wide variety of applications and devices with different capabilities by only loading those components which are necessary in each case. When started, the policy service has a reference to its root domain and only recognizes the import command that can load new classes. Typically, the classes loaded are factories that permit the creation of new managed objects in domains and the first class to be loaded is the factory for the domain objects themselves (see Figure 3.9a below). This enables the policy service to create new domain objects to form a hierarchy of domains under the root domain. Additional, factory objects are then loaded in order to communicate with external event buses, create policies and create adapters for the various sensors and devices in the SMC. The event factory is specific to the event bus used and encapsulates the protocols necessary to communicate with it. However, multiple event factory objects can be created, allowing the policy service to connect to different event buses with different underlying protocols e.g. XMLBlaster, Siena. Similarly, new types of policies e.g., delegation, filtering, etc. can be defined by providing and dynamically loading the corresponding factory. Adapters can also be created for interacting with specific sensors in a pervasive computing environment. For example a *bsn* factory object encapsulates the code for inter-

acting using IEEE 802.15.4 radio with BSN sensor nodes[4] that are used for eHealth monitoring. Specific factories can then be defined for each of the different types of BSN sensors in use eg. hr sensor for heart-rate monitoring which uses the basic bsn adapter for interaction with BSN nodes.

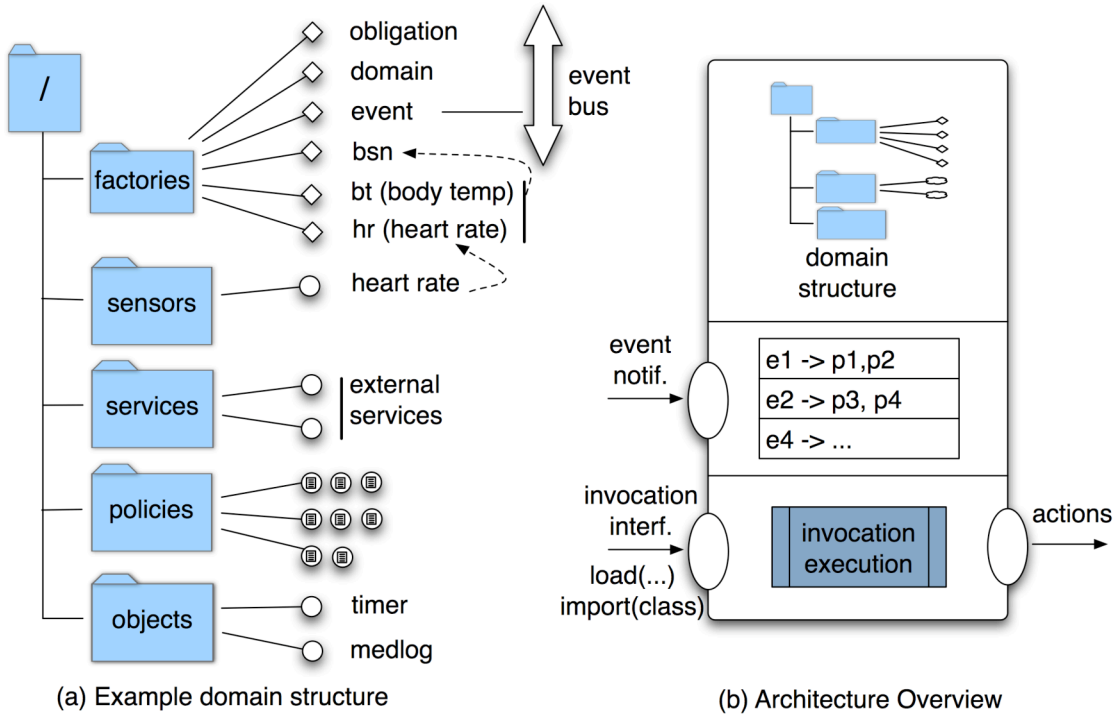

Figure 3.9 Policy Interpreter architecture and domain organisation

As shown in Figure b the overall architecture of the policy service comprises the domain structure, the table matching obligation policies to events and the execution invocation engine which is used to make the calls to the objects inside the domain structure. Conceptually the policy service has an event interface through which event notifications are received, an invocation interface through which external invocations are received and an action interface through which calls are made to external objects.

XML Programmability

Although we are planning to provide a higher-level declarative language for the specification of policies, Ponder2 currently uses XML in its internal representation as well as its input. Ponder2 interprets XML as a sequence of statements that identify managed object and sub-elements of that XML (typically commands with arguments) that are to be sent to those managed objects. For example the following snippet identifies the root domain ("/") and sends it an add command. The add command has its own structure and information saying what is to be added. From the snippet we can see that the managed object to be added to the root domain will be called *newobject*.

```
<use name="/">
 <add name="newobject">
  ...
 </add>
```

[4] These BSNs were developed in the DTI UbiMon project (see http://www.ubimon.org). They have very low power 16-bit processors, 64 KB RAM, 256 KB Flash memory, 6 analog channels for sensors and communicate using IEEE 802.15.4 radio

```
</use>
```

Event types are defined by executing a command on the appropriate event factory and policy instances are similarly created by using the policy factory as shown below.

```
<use name="/Event">
 <add name="hrGT140">
  <use name="/Template/event">
   <create>
    <arg name="hrvalue"/>
    <arg name="time"/>
   </create>
  </use>
</use>
```

```
<use name="/Policy">
 <add name="obliggt65">
  <use name="/Template/policy">
   <create type="obligation" event="/Event/hrGT140"
active="true">
            <arg name="hrvalue"/>
            <arg name="time"/>
            <action>
            <use name="/warning_alarm" alarm="on"/>
            </action>
   </create>
  </use>
 </add>
</use>
```

Communications Library

Ponder2 uses a communications library that can support multiple protocols for remote communciations. These protocols are loaded dynamically as required. The interpreter knows nothing about the protocols until it comes across a URL containing an unknown protocol when a remote object is specified. At that time Ponder2 searches to see if an appropriate protocol module exists, if so it is loaded and used. New protocols are relatively simple to write and can be added to an interpreter by including the jar file containing the implementation of the protocol in the interpreter's classpath. Currently provided protocols include RMI and SOAP/Web-service communications.

Interacting with the interpreter

The policy service can be invoked through an RMI interface, a web-service interface or through a command line interface that resembles the Bourne shell. The latter provides commands for navigation of the domains structure in a similar way to the navigation of a file system in UNIX. Additionally it gives the ability to invoke commands upon the managed objects present in the domain hierarchy and accepts XML directives as input, which are then sent directly to the interpreter.

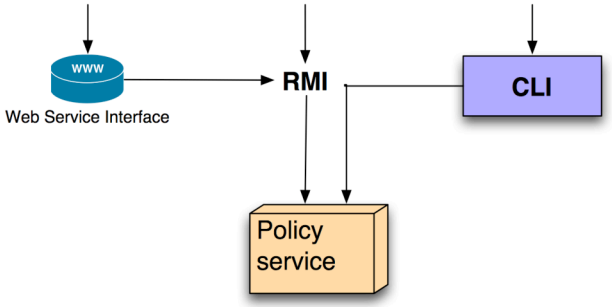

Figure 3.10 Interacting with the Policy Service

3.3.2 Detailed list of extensions planed under the support of EMANICS

- Packaging and deployment for open source release under LGPL
- Higher-level policy-language specification

- Adaptations required for integration with software from other EMANICS partners
- Support and maintenance of the Ponder2 system.
- Integration with access control and authorisation policy framework being currently developed in other projects.

3.3.3 Expected impact

Conceptually, the software is based upon the previous Ponder framework which has had several thousand downloads and for which we continue to receive requests more than one year after it has been withdrawn from distribution.

The software will be integrated in the frameworks resulting from 2 EU projects: TrustCoM (security and contract management in Virtual Organisations) and Diadem Firewall on (security management, distributed firewall management and intrusion response) as well as project funded by the UK-EPSRC. A number of organisations from both industry and academia have already expressed an interest in using this software.

In addition to the traditional network management applications to which the previous version of Ponder was aimed, this new version aims to provide a platform for providing adaptation in pervasive systems and building management applications for pervasive systems. The software is currently being used in order to develop platforms that range from Personal Area Networks for e-Health Monitoring to Autonomous Vehicles and Virtual Organisations.

3.3.4 Progress report

The first version of Ponder2 has been extended and packaged for public release together with a comprehensive set of documentation and programming examples. Ponder2 is available from www.ponder2.net. Although this software has just been released we anticipate that it will form the basis for future collaborations with other EMANICS and external partners.

3.4 OSIMIS porting

3.4.1 Software description

OSIMIS is an object-oriented management platform based on the OSI model [8] and implemented mainly in C++ [4]. It provides an environment for the development of OSI-based management applications which hides the details of the underlying management service/protocol (CMIS/CMIP [9], [10]) through object-oriented Application Program Interfaces (APIs) and allows designers and implementors to concentrate on the intelligence to be built into management applications rather than the mechanics of management service/protocol access. OSIMIS was designed from the beginning with the intent to support the integration of existing systems with either proprietary management facilities or different management models. As such, it also supports a generic CMIS/P to SNMP application gateway [6] for managing SNMP-capable devices. Different methods for the interaction with real managed resources are supported, encompassing loosely coupled resources as is the case with subordinate agents and management hierarchies.

OSIMIS was originally developed in a number of European research projects, namely RACE NEMESYS and ICM and ESPRIT MIDAS and PROOF. It has been used extensively in both research and commercial environments and has served as the management platform for a number of other research projects (RACE, ESPRIT, ACTS, FP6). It

has been used extensively for research and in the 1990's it was being used by many research institutions worldwide, distributed by University College London (UCL). After its main developer Prof. G. Pavlou moved to the University of Surrey in the beginning of 1998, the software stopped being ported to latest version compilers and Unix systems and was eventually withdrawn from the public domain. The intention is to make it again publicly available in the context of EMANICS.

3.4.1.1 OSIMIS component overview

OSIMIS uses the ISODE (ISO Development Environment) [21] as the underlying OSI upper layer protocol stack. The OSIMIS services and architecture are shown in Figure 3.11. In the layered part, applications are programs while the rest are building blocks realised as libraries. The lower part shows the generic applications provided; from those the ASN.1 and GDMO tools are essential in providing off-line support for the realisation of new MIBs. The thick line indicates all the APIs an application may use. In practice though most applications use only the Generic Managed System (GMS) and the Remote MIB (RMIB) APIs when acting in agent and manager roles respectively, in addition to the Coordination and high-level ASN.1 support APIs. The latter are used by other components in this layered architecture and are orthogonal to them, as such they are shown aside (Figure 3.12). Directory access for address resolution may or may not be used, while the Directory Support Service (DSS) API provides more sophisticated searching and discovery facilities.

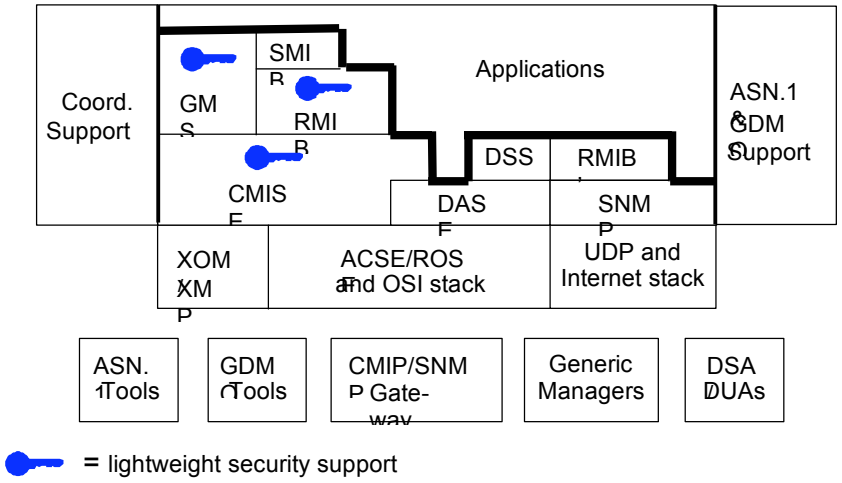

Figure 3.11 OSIMIS layered architecture and generic applications

OSIMIS comprises the following types of support:

- high-level object-oriented APIs realised as libraries,
- tools (compilers/translators) as separate programs supporting the above APIs,
- generic applications such as browsers, gateways, directory servers,
- specific useful management applications.

Some of these services are provided by ISODE and these are:

- the OSI Transport (class 0), Session and Presentation protocols
- the Association Control and Remote Operations Service Elements (ACSE and ROSE),
- the Directory Access Service Element (DASE),
- an ASN.1 compiler with C language bindings (the pepsy tool),
- a full Directory Service implementation including an extensible Directory Service Agent (DSA) and a set of Directory User Agents (DUAs).

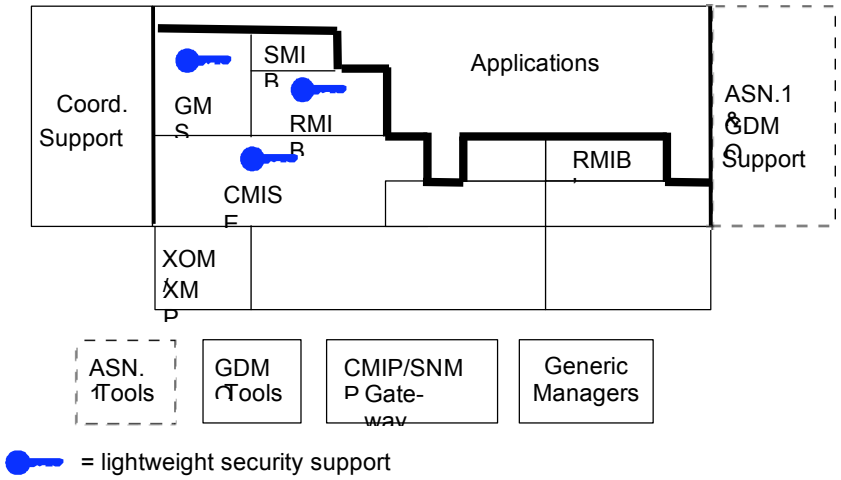

Figure 3.12 The components provided by OSIMIS

OSIMIS is built as an environment using ISODE and is mostly implemented in the C++ programming language. The services it offers are:

- an implementation of CMIS/P using the ISODE ACSE, ROSE and ASN.1 tools,
- high-level ASN.1 support that encapsulates ASN.1 syntaxes in C++ objects,
- an ASN.1 Attribute compiler which uses the ISODE pepsy compiler to automate to a large extent the generation of syntax C++ objects,
- a Coordination mechanism that allows an application to be structured in a fully event-driven fashion and can interwork with similar mechanisms,
- the Generic Managed System (GMS) which is an object-oriented OSI agent engine offering a high-level API to implement new managed object classes, a library of generic attributes, notifications and objects and systems management functions,
- a suite of lightweight security mechanisms, which meet many requirements for access control, authentication, data integrity and data confidentiality,
- a compiler for the OSI Guidelines for the Definition of Managed Objects (GDMO) [12] language which complements the GMS by producing C++ stub managed objects covering every syntactic aspect and leaving only behaviour to be implemented,
- the Remote and Shadow MIB high-level object-oriented manager APIs,
- a Directory Support service offering application addressing and location transparency services,
- a generic CMIS/P to SNMP application gateway driven by a translator between SNMP and OSI GDMO MIBs,
- a set of generic manager applications.

3.4.1.2 Underlying ISODE communication environment

The ISO Development Environment (ISODE) [21] is a platform for the development of OSI services and distributed systems. It provides an upper layer OSI stack that conforms fully to the relevant ISO/ITU-T recommendations and includes tools for ASN.1 manipulation and remote operations stub generation. Two fundamental OSI applications are provided: Directory Service (X.500) [7] and File Transfer (FTAM) implementations. ISODE is implemented in the C programming language [5]. The upper layer protocols realised are the transport, session and presentation protocols of the OSI seven-layer model. The Association Control, Remote Operations and Reliable Transfer application

layer Service Elements (ACSE, ROSE and RTSE) are also provided which, when used in conjunction with the ASN.1 support, act as building blocks for higher level services.

ASN.1 manipulation is very important to OSI distributed applications. The ISODE approach for a programmatic interface (API) relies on a fundamental abstraction known as *Presentation Element* (PE). This is a generic C structure capable of describing in a recursive manner any ASN.1 data type. An ASN.1 compiler known as *pepsy* is provided with C language bindings, which produces concrete representations i.e. C structures corresponding to the ASN.1 types and also encode/decode routines that convert those to PEs and back. The presentation layer converts between PEs and the encoded data stream according to the encoding rules (e.g. BER).

3.4.1.3 Management protocol and abstract syntax support

OSIMIS is based on the OSI management model as the means for end-to-end management and as such it implements the OSI Common Management Information Service/Protocol (CMIS/P). This is implemented as a C library and uses the ISODE ACSE and ROSE service elements together with their ASN.1 support. Every request and response CMIS primitive is realised through a procedure call. Indications and confirmations are realised through a single 'wait' procedure call. Associations are represented as communication end-points (file descriptors) and operating system calls e.g. the Berkeley UNIX `select`(2) can be used for multiplexing them to realise event-driven policies. The OSIMIS CMIS API is known as the MSAP API, standing for Management Service Access Point. OSIMIS also includes an implementation of the Internet SNMPv1 which is used by the generic application gateway between the two. Applications using CMIS need to manipulate ASN.1 types for the CMIS managed object attribute values, actions, error parameters and notifications.

Regarding ASN.1 manipulation, it is up to an application to encode and decode values as this adds to its dynamic nature by allowing late bindings of types to values and graceful handling of error conditions. From a distributed programming point of view this is unacceptable and OSIMIS provides a mechanism to support high-level object-oriented ASN.1 manipulation, shielding the programmer from details and enabling distributed programming using C++ objects as data types. This is achieved by using polymorphism to encapsulate behaviour in the data types determining how encoding and decoding should be performed through an ASN.1 meta-compiler which produces C++ classes for each type. Encode, decode, parse, print and compare methods are produced together with a get-next-element one for multi-valued types (ASN.1 SET OF or SEQUENCE OF). Finally, the very important ANY DEFINED BY construct is automatically supported through a table driven approach, mapping types to syntaxes. This high-level OO ASN.1 approach is used by higher level OSIMIS APIs.

3.4.1.4 Application coordination support

OSIMIS provides an object-oriented infrastructure in C++ that allows an application to be organised in a fully event-driven fashion under a single-threaded execution paradigm, where every external or internal event is serialised and taken to completion on a 'first-come-first-served' basis. This mechanism allows the easy integration of additional external sources of input or timer alarms and it is realised by two C++ classes: the *Coordinator* and the *Knowledge Source* (KS). There should always be one instance of the Coordinator or any derived class in the application, whilst the KS is an abstract class that facilitates usage of the coordinator services and integrates external sources of input and timer alarms. All external events and timer alarms are controlled by the coordinator

whose presence is transparent to implementors of specific KSs through the abstract KS interface.

This coordination mechanism is designed in such a way as to allow integration with those provided by other systems; which is achieved through special classes derived from the coordinator, allowing interworking with a particular mechanism. These specialised coordinator classes still control the sources of input and timer alarms of the OSIMIS KSs, but pass on the task of performing the central listening to the other system's coordination mechanism. This is required for OSIMIS agents which wish to receive association requests, since ISODE imposes its own listening mechanism which hides the Presentation Service Access Point (PSAP) on which new ACSE associations are accepted. A similar mechanism is needed for Graphical User Interface technologies which have their own coordination mechanisms: In which case, a new specialised coordinator class is needed for each of them. The X-Windows Motif, the Tcl/Tk interpreted language and the InterViews graphical object library are fully integrated.

3.4.1.5 The generic managed system

The Generic Managed System (GMS) provides support for building agents that offer the full functionality of the OSI management model, including scoping, filtering, access control, linked replies and cancel-get. OSIMIS fully supports the *Object Management* [14], *Event Reporting* [16] and *Log Control* [17] Systems Management Functions (SMFs); the qualityofServiceAlarm notification of the *Alarm Reporting* [14] SMF; together with profiles of the *Access Control* [20], *Monitor Metric* [18] and *Summarisation* [19] SMFs. In conjunction with the GDMO compiler the GMS offers a very high level API for the integration of new managed object classes where only semantic aspects (behaviour) need to be implemented. It also offers different methods of access to the associated real resources, including proxy mechanisms, based on the Coordination mechanism.

The Generic Managed System is built using the coordination and high-level ASN.1 support infrastructure and most of its facilities are provided by three C++ classes which interact with each other:

- the *CMISAgent*, which provides OSI agent facilities,
- the *MO* which is the abstract class providing generic managed object support,
- the *MOClassInfo* which is a meta-class for a managed object class.

The GMS library also contains generic attribute types such as counter, gauge, counterThreshold, gaugeThreshold and tideMark, and specific attributes and objects as in the Definition of Management Information (DMI) [12], which relate to the SMFs. The object-oriented internal structure of a managed system built using the GMS in terms of interacting object instances is shown in Figure 3.

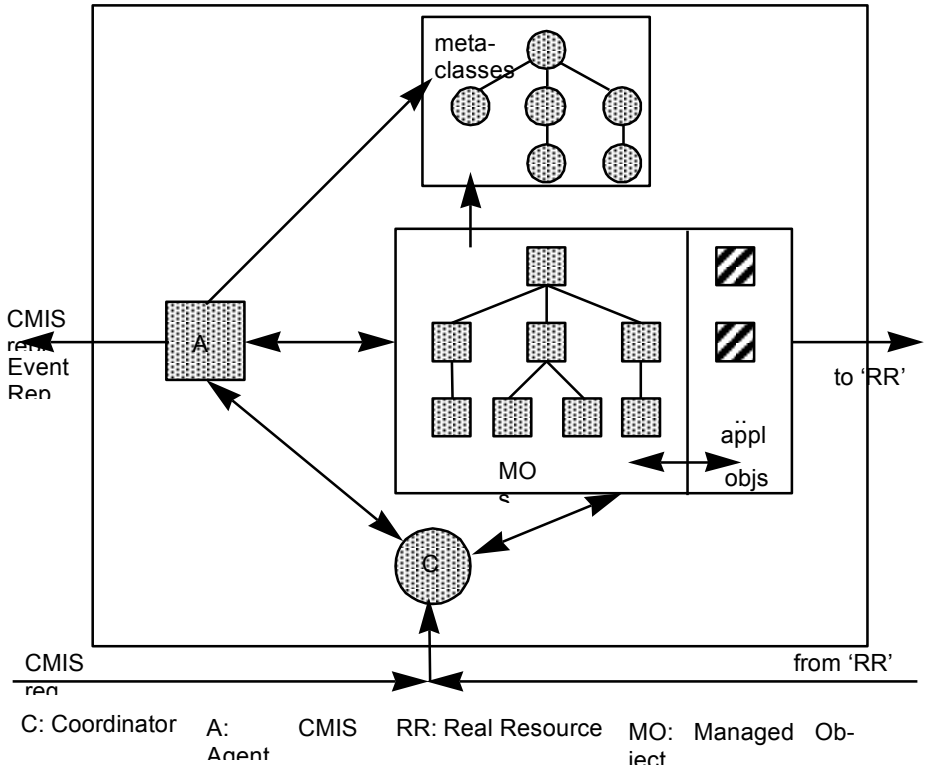

Figure 3.13 The GMS object-oriented architecture

Managed object instances and meta-classes

Every specific managed object class needs access to information common to the class which is independent of all instances and common to all of them. This information concerns attributes, actions and notifications for the class, initial and default attribute values, 'template' ASN.1 objects for manipulating action and notification values, integer tags related to the object identifiers etc. This leads to the introduction of a common meta-class for all the managed object classes, the MOClassInfo. The inheritance tree is internally represented by instances of this class linked in a tree fashion as shown in the 'meta-classes' block of Figure 3.13.

Specific managed object classes are simply realised by equivalent C++ classes produced by the GDMO compiler and augmented manually with behaviour. Through access to meta-class information requests are first checked for correctness and authorisation before the behaviour code that interacts with the real resource is invoked. Behaviour is implemented through a set of polymorphic methods which may be redefined to model the associated real resource. Managed object instances are linked internally in a tree mirroring the containment relationships - see the "MOs" part of Figure 3. Scoping becomes simply a tree search, whilst special care is taken to make sure the tree reflects the state of the associated resources before scoping, filtering and other access operations. Filtering is provided through compare methods of the attributes which are simply the C++ syntax objects, or derived classes (when behaviour is coded at the attribute level.)

Real resource access

There are three possible types of interaction between the managed object and the associated resource with respect to CMIS Get requests:

- access upon external request,

- 'cache-ahead' through periodic polling of the real resource,
- update through asynchronous event reports from the real resource.

The first one means that no real resource access is performed until a managing process accesses the managed object. In the second, requests are responded quickly, especially with respect to loosely coupled resources, but timeliness of information may be slightly affected. Finally the third one is good but only if it can be tailored so that there is no unnecessary overhead when the agent is idle.

The GMS offers support for all methods through the coordination mechanism. When asynchronous reports or results are required, it is likely that a separate object will be needed to demultiplex the incoming information and deliver it to the appropriate managed object instance. It should be noted here that an asynchronous interface to real resources driven by external CMIS requests is not currently supported as this requires an internal asynchronous interface between the agent and the managed objects. These objects are usually referred to as Internal Communications Controllers (ICCs) and are essentially specialised knowledge source objects.

Systems management functions

As already stated, OSIMIS supports the most important of the systems management functions. As far as the GMS is concerned, these functions are realised as special managed objects and generic attribute and notification types which can be simply instantiated or invoked. This is the case, for example, with the alarm reporting, metric and summarisation objects. In other cases, the GMS knows the semantics of these classes and uses them accordingly e.g. in access control, event and log control. Notifications can be emitted through a special method call and all the subsequent notification processing is carried out by the GMS in a fashion transparent to application code. In the case of the object management SMF the code generated by the GDMO compiler together with the GMS completely hiding the emission of object creation and deletion notifications, as well as the attribute change one when something is changed through CMIS. Log control is realised simply through managed object persistency which is a general property of all OSIMIS managed objects. This is implemented using the GNU version of the UNIX DBM database management system and relies on object instance encoding using ASN.1 and the OSI Basic Encoding Rules to serialise the attribute values. Any object can be persistent so that its values are retained between different incarnations of an agent application. At start-up time, an agent looks for any logs or other persistent objects and simply arranges its management information tree accordingly.

Monitoring and summarisation SMFs

The OSI Structure and Definition of Management Information (SMI/DMI) specify generic attribute types such as counter, gauge, threshold and tide-mark. Gauges model entities with associated semantics e.g. number of calls, users, quality of service etc. or the rate of change of associated counters e.g. bytes per second. Thresholds and tide-marks may be applied to gauges and generate QoS alarms and also attribute value changes, indicating change of the high or low 'water mark'. Such activities are of a *managing* nature. Although thresholding functions could be made part of managed objects modelling real resources, it does not take long to recognise their genericity. As such, they are better provided elsewhere so that they become re-usable. The relevant ISO/ITU-T group recognised the importance of generic monitoring facilities and standardised the Metric Monitor [17] and Summarisation [18] systems management functions. By making such functions generic, it is possible to implement them once and make them part of the associated platform infrastructure.

The whole idea behind monitor metric objects is to provide thresholding facilities in a *generic* fashion. Monitor metric objects may be instantiated within an application in an agent role and be configured to monitor, at periodic intervals, an attribute of another real resource managed object. That attribute may belong to a network element managed object but also to a higher-level management application, being mapped onto lower-level managed objects through an Information Conversion Function (ICF). The observed attribute should be a counter or gauge and the metric object either observes it 'as is' or converts the observed values to a rate (derived gauge) over time. Statistical smoothing of the observed values is also possible if desired.

The main importance of this facility is the attachment of gauge thresholds and tide-marks to the resulting derived gauge which may generate quality of service alarms and indicate the high and/or low 'water mark', as desired by systems using this function. In fact, the metric objects essentially enhance the 'raw' information model of the observed object. The metric monitor functionality can be summarised as:

- *data capture:* through observation or 'scanning' of a managed object attribute,
- *data conversion*: potential conversion of a counter or gauge to a derived gauge,
- *data enhancement:* potential statistical smoothing of the derived result,
- *notification generation:* QoS alarm and attribute value change notifications.

The metric monitoring model is shown in Figure 3.14.

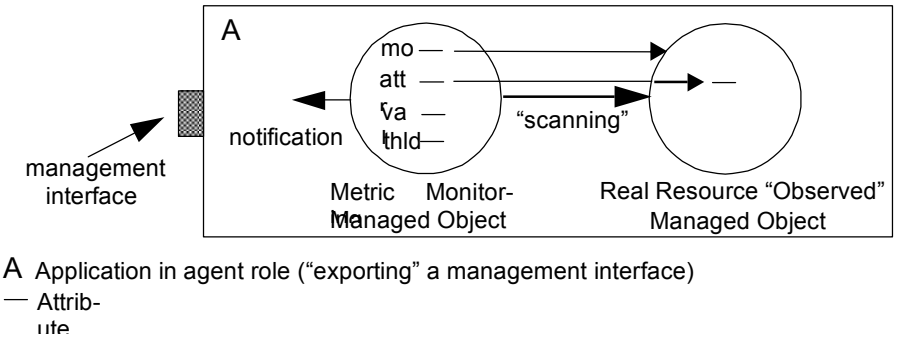

Figure 3.14 The metric monitoring model

The metric objects offer the OSI management power through event reporting and logging, even if the 'raw' observed management information model does not support such notifications. More importantly, they obviate the use of rates, thresholds and tide-marks in a way tied to specific managed object classes but they allow the same flexibility and power dynamically, whenever a managing system needs it. Such a monitoring facility reduces the management traffic between applications and their impact on the managed network by supporting an event-based operation paradigm.

The summarisation objects extend the idea of monitoring a single attribute to monitoring many attributes across a number of selected managed object instances. They offer similar but complementary facilities to metric objects. In this case, there are no comparisons or thresholding but only the potential statistical smoothing and simple algorithmic results of the observed values (min, max, mean and variance). These are reported periodically to the interested managing systems through notifications. The observed managed objects and attributes can be specified either by supplying explicitly their names or through CMIS scoping and filtering. The observed values may be raw ones, modelling an underlying resource, or enhanced values as observed by metric objects. They can

be reported either at every observation period or after a number of observation periods (buffered scanning).

3.4.1.6 Generic high-level manager support

The Remote MIB (RMIB)

The Remote MIB (RMIB) support service offers a higher level API which provides the abstraction of a proxy agent object. This handles association establishment and release; hides object identifiers through friendly names; hides ASN.1 manipulation using the high-level ASN.1 support; hides the complexity of CMIS distinguished names and filters through a string-based notation; assembles linked replies; provides a high level interface to event reporting which hides the manipulation of event discriminators and finally provides error handling at different levels. There is also a low level interface for applications that do not want this friendliness and the performance cost it entails but they still need the high-level mechanisms for event reporting and linked replies.

In the RMIB API there are two basic C++ classes involved: the *RMIBAgent* which is essentially the proxy object (a specialised KS in OSIMIS terms) and the *RMIBManager* abstract class which provides call-backs for asynchronous services offered by the RMIBAgent. While event reports are inherently asynchronous, manager to agent requests can be both: synchronous, in an RPC like fashion, or asynchronous. In the latter case linked replies could all be assembled first or passed to the specialised RMIBManager one by one. It should be noted that in the case of the synchronous API the whole application blocks until the results and/or errors are received, or a timeout occurs, whilst this is not the case with the asynchronous API. The introduction of threads or co-routines will obviate the use of the asynchronous API for reasons other than event reporting or a one-by-one delivery mechanism for linked replies.

The Tcl-RMIB API

Being both interpreted and string based, the Tcl language was selected to form the basis for the OSIMIS scripting language *Tcl-RMIB*. The presence of existing string based interfaces to CMIS, via the RMIB, has greatly assisted in the development of this new interface. Tcl-RMIB [10] assists rapid prototyping of management GUIs, event monitors, MIB browsers etc. with similar functionality to that of the RMIB API, supporting both synchronous and asynchronous interactions.

3.4.1.7 The ASN.1 compilers

For each MOC attribute type's syntax, which is specified in ASN.1, the existing ISODE *Pepsy* compiler can produce a C language structure, together with functions for encoding and decoding, printing, string parsing and performing comparisons. High-level C++ abstractions are then used to provide the support for dealing with these ASN.1 based attribute types, in such a way that utilisation of the actual abstract and transfer syntaxes of the ISODE ASN.1 support service are hidden from the implementer by *encapsulating* the Pepsy compiler output. An ASN.1 Attribute compiler can automate the production of the required C++ attribute classes.

3.4.1.8 The GDMO compiler

Being script driven, the OSIMIS GDMO compiler is completely platform-independent i.e. it does not contain any hard-coded knowledge of a specific network management platform, in this case of OSIMIS. Typically a compiler parses a program (of GDMO statements, in this case) and builds up a symbol table representation of the program; in our

compiler the symbol table is represented as a set of C++ objects. The code generation phase of the compiler then produces code for the target platform. In our compiler, the code generation phase is controlled by an interpreted script language. The script language is able to address the information in the compiler's symbol table and to output the information to files that conform to the requirements of a particular network management platform; in software engineering terms, this is achieved by adding meta class information to the C++ symbol table objects so that they can be used as variables in the script language. In short, the symbol table data of the compiler is entirely malleable. It is in this sense that the knowledge of a particular network management platform is not hard-coded in the GDMO compiler.

3.4.1.9 Generic managers

There is a class of applications which are semantic-free and these are usually referred to as MIB browsers as they allow one to move around in a management information tree, retrieve and alter attribute values, perform actions and create and delete managed objects. OSIMIS provides a MIB browser with a Graphical User Interface based on the InterViews X-Windows C++ graphical object library. This allows management operations to be applied and also provides a monitoring facility. Its successor, which is to be re-engineered in Tcl/Tk, will also have the capability of receiving event reports and of monitoring objects through event reporting.

In addition OSIMIS provides a set of programs that operate from the command line and realise the full set of CMIS operations. These may be combined together in a 'management shell'. There is also an event sink application that can be used to receive event reports according to specified criteria. Both the MIB browser and these command line programs owe their genericity to the generic CMIS facilities (empty local distinguished name {} for the top MIB object, actualClass and scoping) and the manipulation of the ANY DEFINED BY ASN.1 syntax through a table driven approach.

3.4.1.10 Sample agent

OSIMIS contains a non-standard implementation of an OSI Transport Protocol (TP) MIB. This manages the TP implementation of the ISODE stack and has proved to be very useful in monitoring the activity of ISODE based applications, such as DSAs, MTAs, transport bridges and even management applications themselves. It also includes an agent with an example object modelling the Unix operating system (*uxObj1*) as a trivial example managed object.

3.4.1.11 Distributed processing example

To demonstrate the use of the OSI management model as a powerful paradigm for distributed processing when supported by high-level object-oriented APIs, the OSIMIS platform was extended with a distributed processing example. A *simpleStats* class was specified providing simple statistical services in the form of actions. The currently supported services (actions) are:

- *calcSqrt* - returns the square root of a non-negative real number
- *calcMeanStdDev* - calculates and returns the mean and standard deviation of a series of real numbers

The generic *maction* utility can be deployed to provide a 1 line shell script (!) that provides access to these services. In case a more complex client program needs to utilise these services, the RMIB access API may be used; a demonstration client is included

which required only 20 lines in C++. The OSIMIS location transparency service may be used to hide the location where an instance of *simpleStats* object executes.

3.4.1.12 The generic CMIS/P to SNMP gateway

The industry standard for network element management is the Internet SNMP, which is less powerful than the OSI CMIP. The same holds for the relevant information models; the OSI is fully object-oriented while the SNMP supports a simple remote debugging paradigm. Generic application gateways between them are possible without any semantic loss for conversion from CMIS/P to SNMP as the latter's operations and information model are a subset of the OSI ones. Work for standards in this area has been driven by the Network Management Forum (NMF) while the ICM project contributed actively to them and also built a generic application gateway.

This work involves a translator between Internet MIBs to equivalent GDMO ones (i.e. the `imibtool`) and a special script for the GDMO compiler which produces run-time support for the generic gateway. That way the handling of any current or future MIBs will be possible without the need to change a single line of code. It should be added that the generic gateway works with SNMPv1 but may be extended to cover SNMPv2.

A fundamental design requirement for the gateway is to achieve seamless interoperability between TMN management Operations Systems (OS) or Mediation Functions (MF) and SNMPv1 managed resources. An efficient mapping is essential given the fact that the gateway introduces an intermediate hop in the manager/agent communication path, see Figure 3.15.

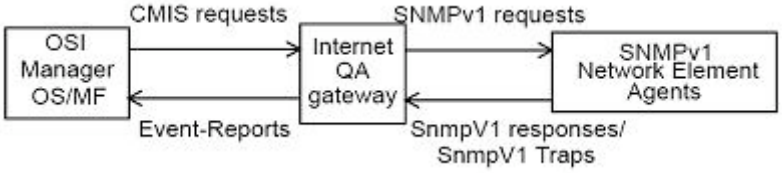

Figure 3.15 Manager/agent communication paths

3.4.2 Detailed List of extensions planed under the support of EMANICS

As already stated, this software is not operational anymore since the late 1990's. In addition, it is not available anymore in the public domain. So the key extension is to port it to the latest version C/C++ compilers and to make it publicly available as an example OSI management implementation and platform for building applications. The last publicly available version was running on Solaris and used an early version of the GNU compilers. Earlier versions had also run on SunOS and HPUX. As such, the key targets are the following:

- Port it to the latest version GNU C/C++ compilers under Linux and Solaris.
- Make publicly available both OSIMIS and ISODE under a Lesser General Public Licence (LGPL) through a Web site that will be created.

The key target will be to re-instate the core parts of the infrastructure that include generic agent and manager infrastructure and key agent applications. Additional developments will target the addition of new important infrastructure. The detailed list of tasks is the following.

- Task 1: Port the ISODE environment to the latest compiler versions on Linux and Solaris, this will be made available separately.

- Task 2: Port the OSIMIS core components to the latest compiler versions on Linux and Solaris. These will include the ASN.1/GDMO compilers, the generic agent infrastructure, the example agents (unix MIB, OSI transport protocol MIB, example distributed processing MIB) and the generic command line managers.
- Task 3: Port additional parts of the infrastructure that have not been ported for a long time. These will include the generic CMIS/P to SNMP gateway, the Tcl-CMIS manager API and possibly the Motif/InterViews-based generic MIB browser.
- Task 4: Distribute a beta version to EMANICS partners to be tried and tested.
- Task 5: Make both OSIMIS and ISODE software available through relevant Web pages which will be constructed.
- Task 6: Build a native agent for (parts of) the GDMO version of TCP/IP MIB-II as in RFC1214 as a realistic specific agent for experiments.

3.4.3 Expected impact

OSI-SM is taught in many Network Management courses and the software is expected to be used in many Universities for teaching / demonstration purposes. In addition, given that OSI-SM is still used in telecommunications environments, it is likely that developers will acquire it. Given its previous popularity, it should enhance the EMANICS visibility. Finally, it will be used by EMANICS partners for additional experimentation and comparisons.

It should be finally added that the software was initially developed in the EU ESPRIT INCA, PROOF and MIDAS projects and the RACE NEMESYS and ICM ones. Being a key result of European research, EMANICS provides an excellent opportunity to also expose to the wider community previous European project results.

3.4.4 Progress report

We have at this point managed to port the majority of ISODE and OSIMIS to Linux FedoraCore 4 with GNU gcc and g++ 4.0.2. We also did port them to a late version of Solaris with gcc/g++ 3.6. As such, we now have at Surrey a working version of OSIMIS with the basic agents and all the command line generic managers working. The only current problem is that the automatically produced pretty-print routine for new ASN.1 syntaxes seems to crash and it has been disabled - as such, it needs to be hand-written; this should be fixed in the future. This version is about to be packaged and passed to UPC to be beta tested. The next step will be to port the Tcl-CMIS manager interface and the generic CMISP to SNMP gateway. Then a full distribution will be made available within EMANICS first and eventually to the wider community.

4 Summary and Conclusions

The activity in WP6 started immediately after the kickoff meeting in late January. Two tasks were launched, one on open source packages inventory and one in the support of EMANICS-brewed Open Source packages. For both tasks the progress is measurable and concrete results are available to the public : online inventory database, 4 EMANICS Open Source projects, out of which 2 already integrate the supported developments in their public releases.

The delay of 6 weeks that occurred at the beginning of the project and which has led to the delayed delivery of this deliverable, is due to the necessity of the prior vote and validation by the general assembly (held 6 weeks after the beginning of the project) of the funding model and funding amount for the Open Source activity. This delay has now been compensated by the development teams in the various supported developments.

The activities started in WP6 have been proven to be a great success so far both in terms of effective extension/improvement/visibility of the supported open source package (the availability of Ponder 2 has just been announced world wide on 30/10/2006) and as a fantastic vector for integration in EMANICS. Based on this success, a second call for applications will be launched in early December 2006 with funding allocated in the first weeks of January 2007. Several additional open-source components developed in 2006 within EMANICS already represent very promising candidates.

5 References

[1] The EMANICS Web site. http://www. EMANICS.org

[2] The Simple-Web. http://www.simple-web.org

[3] Libre Source. http://libresource.inria.fr

[4] B.Stroustrup, “The C++ Programming Language,” Addison-Wesley, Reading, MA, 1986.

[5] B.W.Kernigham, D.M.Ritchie, “The C Programming Language,” Prentice-Hall, New Jersey, 1978.

[6] K.McCarthy, G.Pavlou, S.Bhatti, J.Neuman De Souza, “Exploiting the Power of OSI Management for the Control of SNMP-capable Resources Using Generic Application Level Gateways,” ISINM’95.

[7] ITU-T X.500, Information Processing - Open Systems Interconnection - The Directory: Overview of Concepts, Models and Service, 1988.

[8] ITU-T X.701, Information Technology - Open Systems Interconnection - Systems Management Overview, July 1991.

[9] ITU-T X.710, Information Technology - Open Systems Interconnection - Common Management Information Service Definition, Version 2, July 1991.

[10] ITU-T X.711, Information Technology - Open Systems Interconnection - Common Management Information Protocol Specification, Version 2, 7/91.

[11] T.Tin, G.Pavlou, R.Shi, “Tcl-MCMIS: Interpreted Management Access Facilities,” Proc. of the 6th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, Ottawa, Canada, October 1995.

[12] ITU-T X.721, Information Technology - Open Systems Interconnection - Structure of Management Information - Part 2: Definition of Management Information, February 1992.

[13] ITU-T X.722, Information Technology - Open Systems Interconnection-Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects, August 1991.

[14] ITU-T X.730, Information Technology - Open Systems Interconnection - Systems Management: Object Management Function, January 1992.

[15] ITU-T X.733, Information Technology - Open Systems Interconnection - Systems Management: Alarm Reporting Function, February 1992.

[16] ITU-T X.734, Information Technology - Open Systems Interconnection - Systems Management: Event Management Function, February 1992.

[17] ITU-T X.735, Information Technology - Open Systems Interconnection - Systems Management: Log Control Function, September 1992.

[18] ITU-T X.738, Information Technology - Open Systems Interconnection - Systems Management: Metric Objects and Attributes, 1994.

[19] ITU-T X.739, Information Technology - Open Systems Interconnection - Systems Management: Summarisation Function, 1994.

[20] ITU-T X.741, Information Technology - Open Systems Interconnection - Systems Management: Objects and attributes for access control, 1995.

[21] M.T. Rose, J.P. Onions, C.J. Robbins, “The ISO Development Environment User’s Manual Version 7.0” PSI Inc / X-Tel Services Ltd., July 1991.

[22] [1] M. Burgess, "A Site Configuration Engine", Computing Systems, 8(3), 1995

[23] [2] J. Schoenwaelder, "Specific Simple Network Management Tools", Proc. LISA XV, December 2001

Abbreviations

AAA	Authentication, Authorization and Accounting
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation Number One
BEEP	Bloc Extensible Exchange Protocol
BSD	Berkeley software distribution
CSS	Cascading Style Sheets
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMS	Content Management System
DMI	Definition of Management Information
DOM	Document Object Model
GDMO	Guidelines for the Definition of Managed Objects
GMS	Generic Management System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secured
ISO	International Standards Organization
ITU	International Telecommunications Union
LGPL	Lesser general Public License
MIB	Management Information Base
MOME	Monitoring and Measurment Cluster
NoE	Network of Excellence
OSI	Open Systems Inteconnection
SMC	Self Managed Cell
SMF	System Management Function
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSH	Secure Shell

URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformation

6 Acknowledgement

This deliverable was made possible due to the large and open help of the Work Package 6 team of the EMANICS NoE. Many thanks to all of them.

7 Appendix A : Form for the initial call for Open Source development support

EMANICS Work Package 6 : Open Source Software Initiatives
Proposal Sheet (to be filled & sent to the WP Leader : Olivier Festor + on WP6 mailing list before march 12, 2006)

1. Overall Open Source Software Description
(A 1/2 to 3/4 page description of the concerned software, its current status, its visbility, its use, ... In case of a non existing soft the emphasis should be made on its need and its impact on the community)

2. Licensing & distribution scheme
(license type & distribution scheme used for the software : GPL, LGPL, QPL, ...; available on a given forge, is it or will it be embedded in a third party software distribution, ..., ...)

3. Detailed List of Extensions Planed under the support of EMANICS
(describe all tasks planed and extensions envisioned as part of this support)

4. Expected Impact
(what new "markets" the enhanced software will conquer, how many distributions are envisioned, where is it going to be integrated, what visibility the NoE can gain through this support ?)

5. Cooperation level
(which parts of the extensions planed come from a cooperation among one or more partners in EMANICS, e.g. X will integrate in his software the algorithm defined by Y).

6. Cost & Requested support
(Expected cost overall + requested support. Cost includes the ressources the partner puts on the development without being supported by the NoE. These efforts must be mesurable at the end of the funding period. Request Support contains the amount of money asked to the NoE. The requested support should precisely specify how it is distributed among salary & equipment.)